The OOF Manual: Version 1.0.8.6

W. Craig Carter
Department of Materials Science and Engineering
Massachussets Institute of Technology

Stephen A. Langer
Information Technology Laboratory
National Institute of Standards and Technology

Edwin R. Fuller Jr.
Materials Science and Engineering Laboratory
National Institute of Standards and Technology

May 26, 2000

Contents

List of Figures

Chapter 1

Introduction

1.1 The OOF Project

OOF[1] is public domain software created at the National Institute of Standards and Technology (NIST) to investigate the properties of microstructures. The microstructure of a material is the (usually) complex ensemble of polycrystalline grains, second phases, cracks, pores, and other features occurring on length scales large compared to atomic sizes.

At the simplest level, OOF is designed to answer questions like, "I know what this material looks like and what it's made of, but I wonder what would happen if I pull on it in different ways?", or "I have a picture of this stuff and I know that different parts expand more than others as the temperature increases — I wonder where the stresses are greatest?"

One approach for investigating microstructural behavior is to reduce the representation of a microstructure to a small number of physical parameters (such as grain size or porosity) and develop a model which depends on them. If this type of reductionist approach is predictive, then such models can be extremely useful. However, when physical properties depend on microstructural details (such as the spatial correlation of crystallite orientation, the shapes and dispersion of second phases, extremes of statistical distributions, or local anisotropies) such data reduction is often difficult or pointless.

OOF takes a non-reductionist, brute force approach, but in a user-friendly way. The user starts with a digitized image of the microstructure and builds a data structure on top of it. *All* the data *plus* any that can be inferred by the user is used. Tools are provided to allow the user to graphically select features in the microstructural image and specify their properties. For OOF, the microstructure is a data structure composed of image and property data.

The idea of basing calculations on images is not new. Edward Garboczi[2] and colleagues at NIST have used this approach to investigate behavior of cements and porous media.

[1] "OOF" stands for "**O**bject **O**riented **F**inite Element". The object oriented design of the program allows it to be easily generalized to include new material types and answer new types of questions.

[2] http://ciks.cbt.nist.gov/garboczi

Researchers at Alcoa have developed finite element models of textured materials. The purpose of creating OOF and distributing it freely is to supply a *generic* tool for calculating microstructure–property relations.

Currently, OOF does thermoelastic mechanics calculations. However, there are plans to extend OOF to other types of physical properties. Documentation is also planned to assist users in programming their own elements to include them into the OOF source. Our ambition is that people will use OOF to do types of research that we have not imagined.

The OOF Project actually consists of two programs. PPM2OOF reads images and assigns material properties to them, and OOF reads the output from PPM2OOF and performs virtual experiments on it. This manual describes only the latter program. OOF can be used independently of PPM2OOF, as long as OOF is provided with a data file in the format described in Section 6.3.

1.2 The OOF Program

OOF is a finite element solver designed to operate on data structures which come from images. The image may be a scanned micrograph of a composite material, a satellite image of the earth, the result of a computer simulation, or a hand drawn sketch. Currently OOF performs microstructural thermoelastic calculations in two dimensions while averaging over the out-of-plane direction with plane stress or plane strain. Future versions of OOF will include other types of problems, such as thermal diffusion and piezoelectricity.

OOF works in tandem with a second program, called PPM2OOF[1], which creates the data file for OOF by graphically combining data from an image[3] with user-specified properties. We will use the term *microstructure* to refer to the image and the associated spatially-correlated properties. PPM2OOF creates the data representing the microstructure; that data is saved in a file—called a *.goof* file ("grid for oof")–which is the starting point for an OOF calculation.

A few examples may clarify the purpose of OOF. OOF can be used to calculate the distribution of stresses in a laminar microstructure taken from a digitized optical micrograph (such as the one in Figure 1.1a). After PPM2OOF assigns material properties to the image, creates a finite element mesh, and writes the mesh to a file (the *.goof* file), OOF can be used to perform a variety of virtual mechanical tests to investigate the behavior of this microstructure. For example, in Figure 1.1 the image with its correlated properties was tested in two uniaxial strain states (the horizontal and vertical edges were each displaced normal to themselves) by the specification of boundary conditions in OOF. The elastic equilibrium solution was numerically calculated. Graphical representations of the solution for two separate components of the stress are illustrated in Figure 1.1. It is clear that that stress parallel to the layers is concentrated in the layers, whereas stress perpendicular to

[3]The format of the graphical image is the *portable pixel map* (`.ppm`, `.pgm`, `.pbm`, or `.pnm`) format developed by Jef Poskanzer. The shareware program *xv* is one of many tools that can convert images into ppm format. *xv* can be obtained from `http://www.trilon.com/xv/`. NIST does not endorse any commercial or shareware programs.

the layers is uniformly distributed.

As a second example, consider a simple bi-material, such as that found in some thermostats, where one material expands more than the other as temperature increases. Suppose you drew the picture in Figure 1.2 and saved a scan of it in some graphics file (or, as we did to create this example, simply used a computer drawing program).

You could import that figure into PPM2OOF and by clicking with the mouse, apply material properties to it and then use those properties to create a mesh, like the one illustrated in Figure 1.3.

If you wanted to know how the material will deform as the temperature increases, you would simply import the mesh into OOF via the .goof file, specify the change in temperature with a few clicks and keystrokes, and calculate a solution with one more click. Figure 1.4 illustrates the elastic energy density in the deformed bi-material.

You might also ask how the shear stress would be distributed if the bi-material were fixed to a rigid substrate (finite in the horizontal direction and either thin (plane stress) or very thick (plane strain) in the out-of-page direction). OOF lets you set the appropriate boundary conditions and solve for the stress, as shown in Figure 1.5.

1.2.1 How to Read this Manual

There's no point in telling you how to read this manual, since you'll do what you want anyway, and we'll never know. But you could adopt one of the following strategies:

Impatient Skip the manual. Start the program by typing `oof`. Refer to Section 1.5 when the interface is confusing, and to Chapter 3 when you need to find a particular command. The hierarchical list of menus in Section 3.0.1 may be more useful than the table of contents.

Pedagogical Go through the examples in Chapter 2. Refer to the Overview in Section 1.5 as necessary.

Cautious Read the whole Introduction and then try the examples.

Obstinate Read the whole manual cover to cover and skip the examples.

1.2.2 A Note on Typography in this Manual

Bold face font is used for specific items in the graphical interface: **Quit Button**.

Sans serif font is used for menu items and the names of variables and function arguments: initialize. Since menus can contain submenus (see Section 1.5.1) a slash (/) is used to separate the names of the submenus, with a leading slash to indicate the main OOF menu. For example, /initialize/file is the file command in the initialize submenu of the main menu, and /bc/fix/both is the both submenu of the fix submenu...

Typewriter font is used for things that you type, or that the program types to you: `oof -grid example.goof`.

Italic font is used when introducing a new *definition* or whenever some point needs to be *particularly* stressed.

1.3 Getting OOF and System Requirements

The graphical interface to OOF runs under X-windows. OOF itself is written in C++ and uses the XForms[4], SparseLib++[5], and editline libraries. It has currently been compiled on SGI computers using the SGI C++ compiler, on Sun Solaris-2 with the Workshop 5.0 compiler, and on DEC Alphas running Linux with the GNU egcs compiler.

OOF can be obtained from the OOF website at

`http://www.ctcms.nist.gov/oof`

At that site you can also sign up for the OOF mailing list.

1.4 Starting OOF

Type `oof [options]` on the Unix command line. The options are:

`-text` Don't use the graphical interface. If `-text` is specified, then graphics windows cannot be opened, and all interaction with OOF will be through the text interface described in Section 6.1.

`-file ⟨filenames⟩` Load and process the specified files of OOF commands. The files have the format described in Section 6.1. If more than one filename is specified, they must be separated by commas without any intervening spaces, *e.g,*

```
oof -file startup.com,churnaway.com,finishup.com
```

Files will be processed in the order given.

`-grid ⟨filename⟩` Load the specified `.goof` file, assumed to contain data written by OOF or PPM2OOF. The file is loaded before any command files specified with the `-file` option.

[4] http://bragg.phys.uwm.edu
[5] http://math.nist.gov/sparselib%2b%2b/

`-log` ⟨`filename`⟩ Save a log of all commands to the given file.

`-quit` Quit. Use this if you want OOF to quit immediately after running files given with the `-file` option. Otherwise, OOF will process the files and then expect interactive input.

`-singleclickfns` Use the old style of executing menu commands with a single mouse click on their names in the Menus, and opening a function window with a double click. The more sensible default behavior is the opposite.

`-help` Print this list of options, and quit.

The first thing that OOF does, even before processing any files specified with the `-file` and `-grid` options, is to look for a file called `.oofrc`, either in the current directory or the users home directory. This file should contain commands as described in Section 6.1. These commands will be processed immediately. The `.oofrc` file is a good place to put the definitions of commonly used macros (see Sections 3.11 and 6.1.6) and any other desired initialization commands.

1.5 Overview of the Graphical Interface

The graphical interfaces[6] in OOF are designed to be an intuitive means to carry out virtual experiments on the microstructure in the `.goof` files. There are two major kinds of interfaces—*Menus* are used to issue commands (*eg,* initialize the grid, apply boundary conditions), and *Drawers* are used to view the results (*eg,* stress, strain). The *main* OOF *menu* is displayed when the program is started. Figure 1.6 illustrates the prototypical Menu graphical interface. Figure 1.7 is a Drawer representing the first invariant of the stresses (the trace of the stress tensor, or the negative of the pressure) for an undercooled (reduced uniform temperature field) microstructure corresponding to Figure 1.1.

Pressing the "Home" key on the keyboard will always bring up the main OOF menu. This is very useful when the screen gets cluttered.

1.5.1 Menus

Menus consist of three main *panels* containing lists of submenus, functions, and variables, as well as a number of buttons.

[6]OOF's interface is built with the XForms Library. More information about XForms can be obtained at `http://bragg.phys.uwm.edu/xforms`.

(sub)Menus panel Clicking (any mouse button) on any of the words in the **Menus panel** brings up another Menu (a *sub*Menu) of the current menu. For instance, clicking on the word bc from the main menu in Figure 1.6, brings up an entirely new Menu which has the same form as the main Menu but is designed for setting boundary conditions.

Functions panel Quickly clicking *twice* with any mouse button on a word in the **Functions** panel performs an action. For instance, double clicking on the function equilibrate invokes the solver to calculate the solution on the current grid subject to the current boundary conditions. If the function takes arguments, the default values of the arguments are used when the name is single-clicked.

Clicking *once* on a word in the **Functions** panel produces an additional *function window*. (See Figures 1.8 and 1.9 for examples.) Function windows are used to set arguments (if the function has arguments) and to provide a quick way to execute frequently used functions (even if the function's menu isn't open). For details, see Section 1.5.3.

Historically, the behavior of single and double clicks was reversed, but having a slow double click accidentally interpreted as two single clicks could be disastrous if each single click actually executed a command. With the current scheme, the worst that can happen is that you have to close an extra window or two. If you really prefer the old unsafe behavior, invoke OOF with the `-singleclickfns` option.

Variables panel Clicking on a word in the **Variables** panel causes a graphical interface to appear at the bottom of the Menu which allows variables to be set. Different kinds of variables are set in different ways. The interfaces are meant to be intuitive, but a few comments are in order:

Keyboard Input: Variable values that you type on the keyboard don't take effect until the **Set** button is clicked or the ⟨*return*⟩ key is pressed. Various emacs-like control characters can be used to edit typed input. For example, `Control-U` clears the input field. See Section 6.2 for details.

Boolean variables, taking the values `True` or `False`, are set by clicking on the buttons labeled **True** and **False**. The new value takes effect immediately.

Enumerated variables take a value from a discrete set. For example, the `preconditioner` variable in the main menu can be one of `diagonal`, `block`, `ILU`, `ICP`, or `none`. Choose the value from the pop-up menu that appears at the bottom of the Menu.

Orientations are a special kind of keyboard variable. They are either given as a single number, which is interpreted as a counterclockwise rotation about the z axis (out of the screen) or as Euler angles. In this case, the three angles are to be typed in square brackets and separated by commas. (*e.g.* `[123, 45, 10]`). A *random* orientation can be generated by using the letter `r` in place of any of the numbers.

File Names can be typed. In addition, they have another button that brings up a file browser, listing the files in the current directory. You can change directories with the **Directory** button or by clicking on a directory name. You can restrict the files listed with the **Pattern** button (a pattern of `*.goof*` will list all `.goof` files). After you use the browser, you still have to click the **Set** button before the new value takes effect.

Character Strings can be enclosed in quotation marks (`"like this"`) but they don't have to be. When the default value of an input string contains spaces or is empty, the quotes will be provided automatically.

Home Button The **Home** button brings back the main OOF menu. It is equivalent to the "Home" key on on the keyboard.

Quit/Back Button On all but the main OOF menu, the **Back** button brings back the previous menu. On the main menu, this button quits the program.

Close Button This button closes the menu. If no other menu is open, the main OOF menu is automatically opened, so that there is always at least one menu on the screen.

Freeze Button Normally, when another menu is opened (either by selecting it in the **Menus** panel or by using the **Home** or **Back** buttons), the current menu is closed. However, if the **Freeze** button is highlighted, the current menu stays open.

Record Button OOF contains a primitive facility for recording and playing back frequently used sequences of commands (*macros*). Click once on the **Record** button to start recording. The **Record** button will turn into a **Stop** button. Execute the commands you want to record. (The commands won't actually be executed; just go through the motions.) Go back to the Menu in which you started recording, and click on the **Stop** button. You will be asked to name the macro you've just defined, and the macro will appear in the **Functions** panel of the current Menu. To delete a macro, define a new macro with the same name but containing no commands. To save and load macros, see Section 3.11.

Note that the commands contained in a macro must actually be in the menus at the time that the macro is defined. That means that macros can't refer to other macros, unless the other macros are defined first. There are OOF commands whose names are derived from user-defined objects (element groups and node groups); these commands also cannot be used in macros until the commands appear in the menus. This can cause problems if macro definitions are read from a file before the grid is loaded.

1.5.2 The Message Window

This is where all messages from OOF appear. These include all the executed commands, error messages, and output from various calculations. The Message Window shows up only after there is something to display, so it may not appear when OOF starts up. If needed, there are **Scroll Bars** along the right and bottom edges. The **Close** button makes the window go away—it will reappear when the next message is written. The **Clear** button removes all messages in the window. The **Save** button writes the contents of the window to a file (but see Section 3.12 for a better way to create a log of your OOF session). The **Font** pop-up menu and **Size** control allow you to change the appearance of the text.

The color of the text in the Message Window indicates the type of message. Blue *commands* indicate what function is being executed. Black *data* is the output from the commands, while brown *information* is output that is incidental to the action of the command. For example, the command /output/energy prints the energy in black, but /equilibrate prints the equilibration time and convergence in brown. *Error* messages are red, naturally.

1.5.3 Function Windows

There are two flavors of *Function Windows*, for functions with and without arguments. Both have the following features:

Function Button This is the large button at the top of the window labeled with the function's name. Clicking on this button executes the function. The arrow at the right side of the button indicates that hitting ⟨*return*⟩ when the mouse is in this window will also execute the function.

Dismiss Button Clicking the button labeled *Dismiss* closes the Function Window without executing the function.

? Button The button with the question mark on it prints a short description of the function to the **Message Window**.

Freeze Button When the **Freeze Button** is highlighted, the Function Window will not disappear after the function is executed. Click on the button to turn it on and off. A frozen function window can be used to execute the same function repeatedly without returning to the **Menu** from whence it came. The **Menu** does not need to stay open.

The following features are only present for functions that take arguments:

Set Default Button Click here to make the current values of the arguments in this window the default arguments. The default arguments are the ones used when you double-click on the function name in the Menu, and the ones displayed when the Function Window is first displayed.

Copy Button This button opens another identical Function Window with identical arguments. Having two function windows allows you to repeatedly execute commands with different arguments.

Arguments Each argument to the function is listed at the bottom of the Function Window. You can enter values for the arguments in almost exactly the same way as you enter values for Variables in the Menus. The difference is that there is no equivalent to the **Set** button, since all argument values are automatically set when the function is executed. The ⟨*tab*⟩ key will let you move from one numerical argument to the next. *Important Note!* Remember that the ⟨*return*⟩ key executes the function—don't accidentally execute the function by hitting ⟨*return*⟩ when you only wanted to go to the next argument.

1.5.4 Drawers

Drawers are windows that show pictures of the mesh and the results of the calculation. (See Figure 1.7.) The generic features of the Drawers are described here—for specific details see Chapter 5.

Drawing Area The **Drawing Area** is the region at the right where the drawing appears. You can change the size of the window with your usual window manager commands if you want a bigger region.

Zoom Button This is the small button marked with a "z" at the upper left corner of the Drawing Area. Click on it with the left mouse button to zoom out (*i.e,* to make the image smaller), and with the right mouse button to zoom in. Click with the middle mouse button to make the image fill the window. *Be careful about zooming in too far!* To scroll the image smoothly, OOF stores a full size copy of it off screen. If you zoom in too far, the off screen copy can fill the memory on your X server. See Section 6.4.

Scroll Bars The **Scroll Bars** above and to the left of the Drawing Area let you move the visible region around if you've zoomed in.

Drawing Selector This is the large rectangle at the top left (marked "Stress Invariant 1" in Figure 1.7). It is a pull down menu that controls what's being drawn. Click and drag the left mouse button on it to bring up the menu. The right mouse button will cycle through the options; the middle button will cycle through them the other way.

Dashboard The **Dashboard** is the blue region with various controls on it. There are many different Dashboards. Different Drawers have different sets of Dashboards. They are all described in Chapter 5.

Dashboard Selector The smaller rectangle on the left below the Drawing Selector is the **Dashboard Selector**. It's another pulldown menu that determines which sets of controls appear in the Dashboard.

Hold Button The **Hold Button** to the right of the Dashboard Selector can be used to inhibit redrawing of the image in the Drawing Area. If you're changing many attributes of a large image, you won't want OOF to redraw the image before you've changed all the attributes. Click on the Hold Button (it will light up yellow), make your changes, and click on the hold button again to tell OOF that it's time to draw.

Close Button This makes the Drawer disappear.

Save Button This saves the visible part of the image as a ppm file.

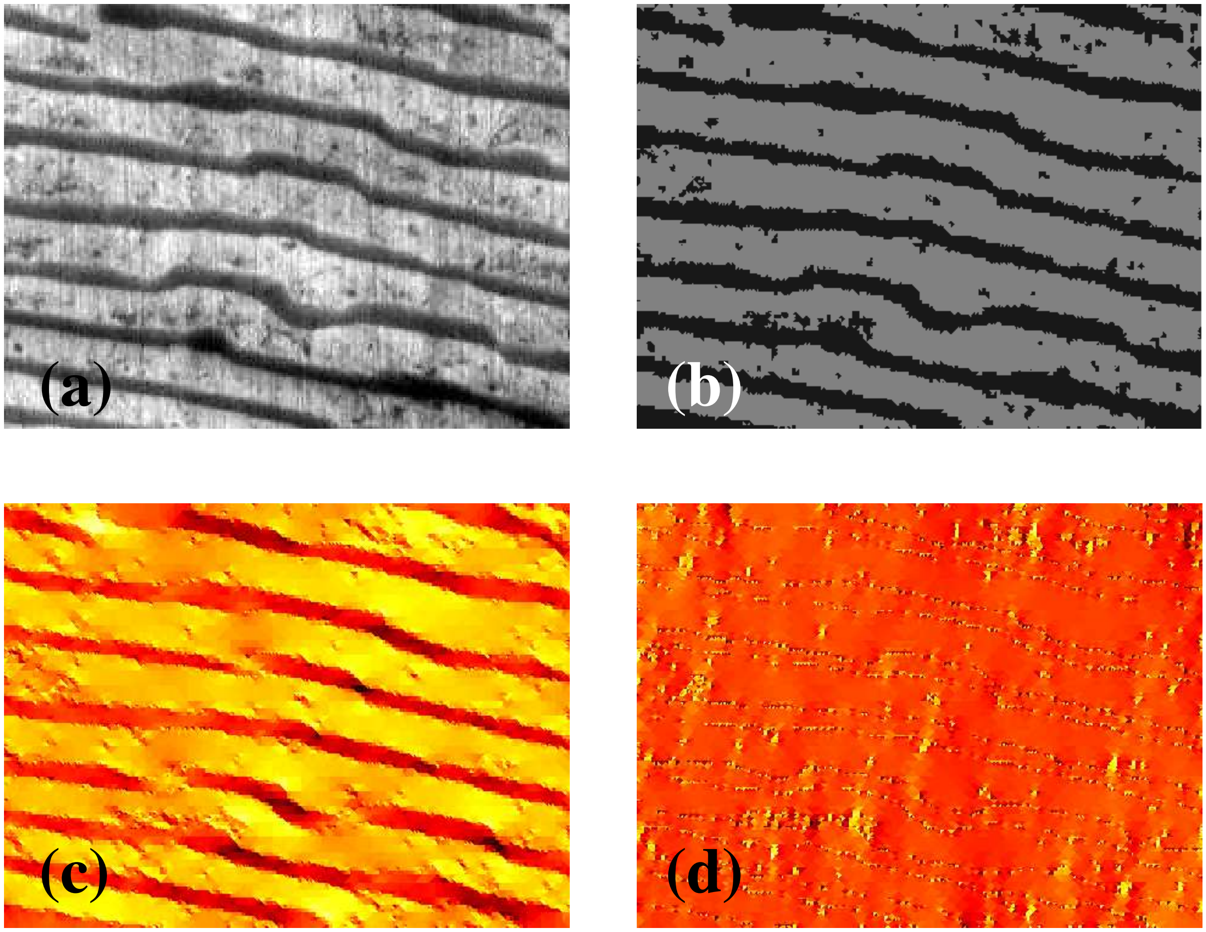

Figure 1.1: **(a)** A 255×190 pixel section of a micrograph of NiO–ZrO_2, courtesy of V. Dravid at Northwestern University.
(b) The finite element grid created from the micrograph by PPM2OOF and ready to be processed by OOF. Elements have been colored dark gray (ZrO_2) and light gray (NiO) to indicate their material type, but the boundaries between elements aren't visible. This non-uniform mesh contains 8916 nodes and 17525 elements. Each phase has cubic symmetry but a different orientation and different thermoelastic coefficients.
(c) The xx component of stress in the sample when subjected to a 1% strain in the x direction, as computed by OOF.
(d) The yy component of stress when the sample is subjected to a 1% strain in the y direction. The difference between (c) and (d) clearly shows the influence of the microstructure. In (c) and (d) dark regions (red, if you have a color version of this manual) are low stress, and light (yellow) regions are high stress.

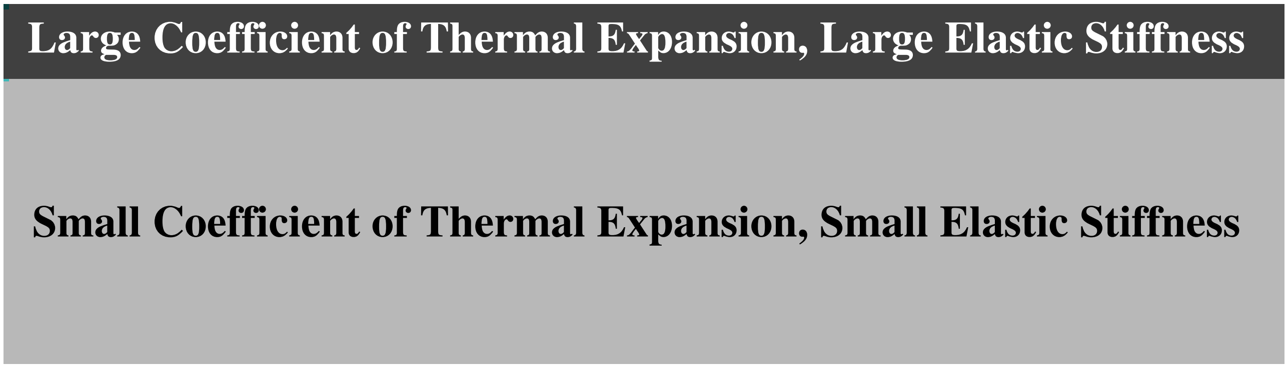

Figure 1.2: Example of a simple drawing and the material properties which will be applied to the drawing.

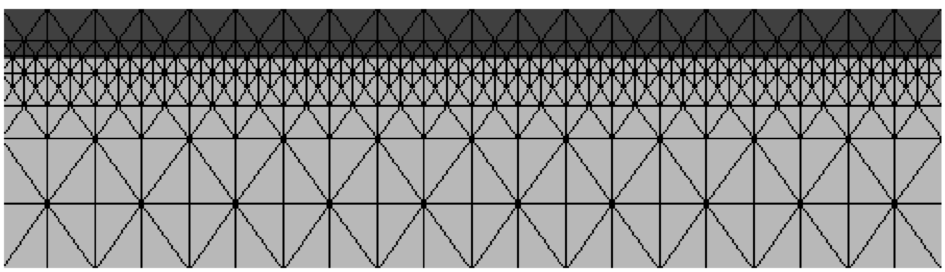

Figure 1.3: An adaptive mesh created by PPM2OOF on the bi-metal strip of Figure 1.2. This mesh is fairly rough and is straightforward to refine.

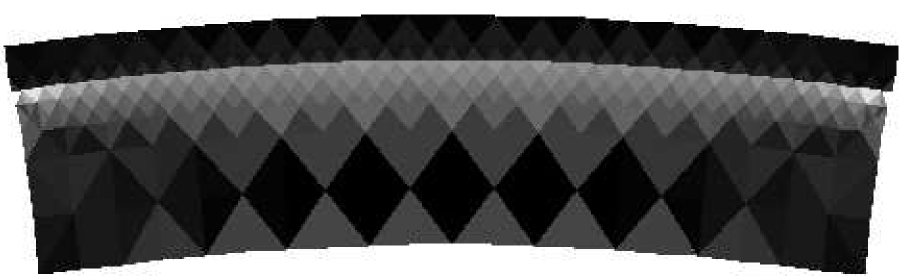

Figure 1.4: Elastic energy density in unconstrained bi-material with differing coefficients of thermal expansion. Dark regions have low energy; light regions have high energy. (Clearly the mesh is too coarse.)

Figure 1.5: Distribution of shear stress for hot bi-material coherently attached to a rigid substrate.

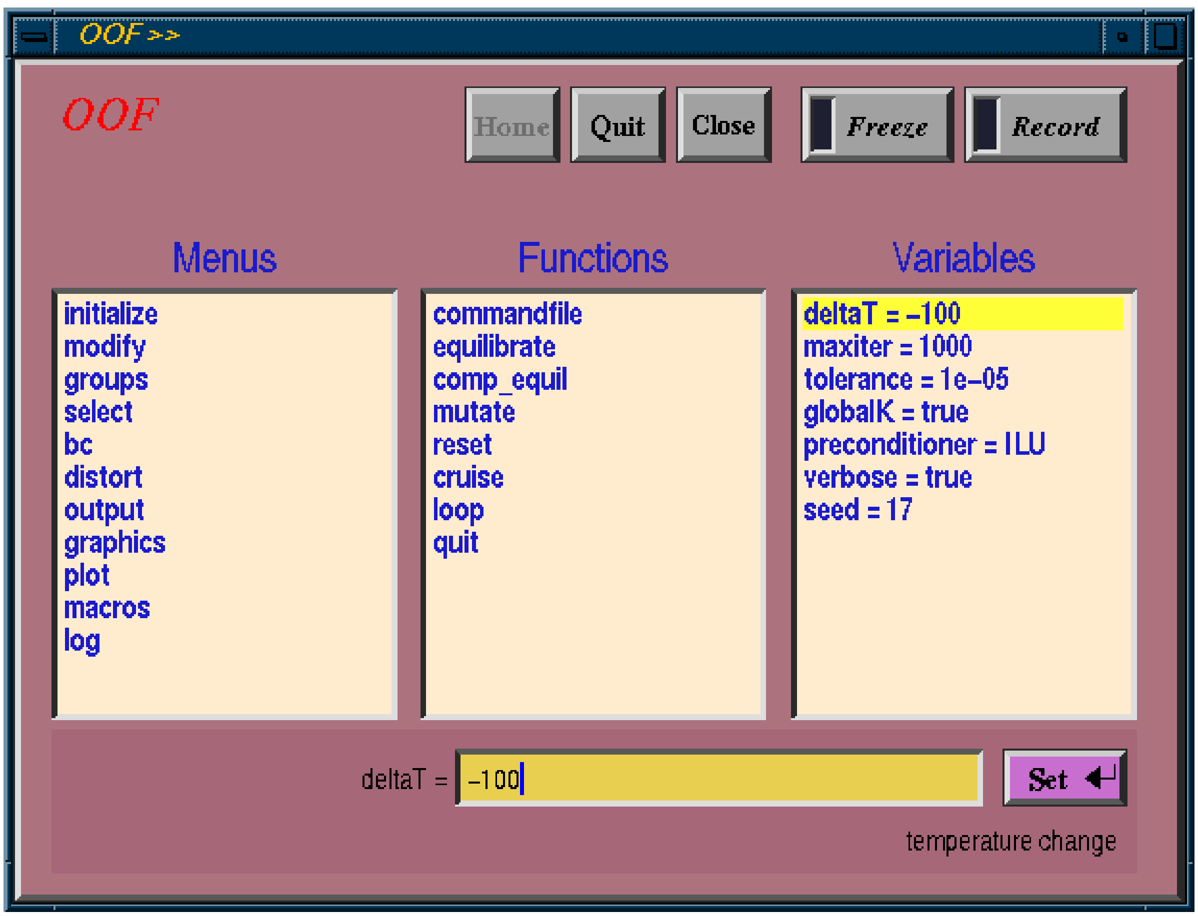

Figure 1.6: The Main OOF Menu. Every other OOF Menu has a nearly identical form: submenus left, functions center, variables left. Clicking on words invokes an action.

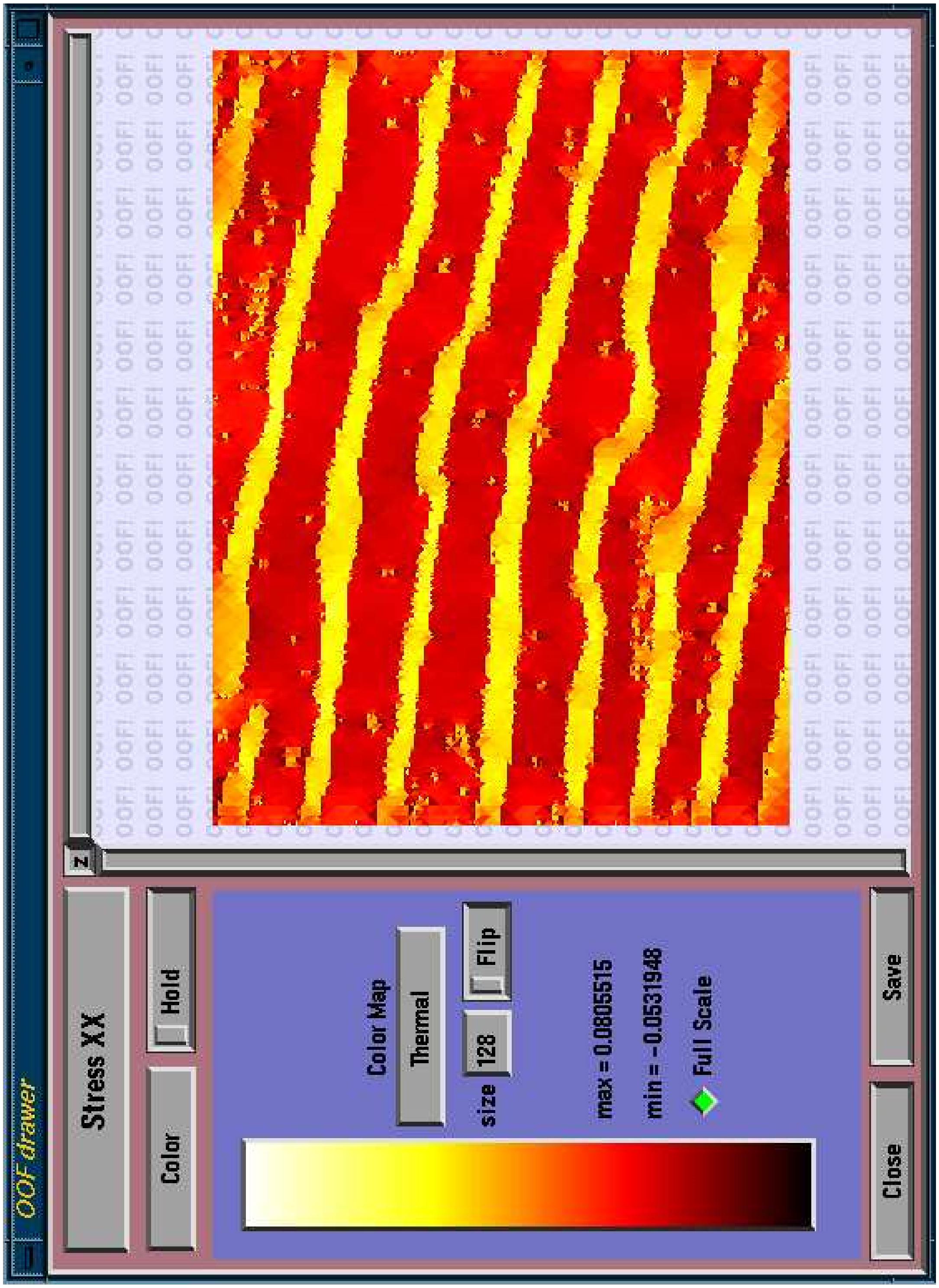

Figure 1.7: Graphical interface (called a *Drawer*) for the stresses due to undercooling the data from Figure 1.1.

Figure 1.8: Function window for a function with no arguments.

Figure 1.9: Function window for a function with arguments.

Chapter 2

Examples

The following examples assume that OOF has been downloaded and can run on your local machine. OOF is started by typing `oof` at a Unix (with X-Windows) prompt.

Typing `oof -help` gives a list of command–line options:

```
\% oof -help
Version 1.0
Usage: oof [options]
Options:   -text            use text interface
           -file <filename> process commands in filename
           -grid <filename> read grid from filename
           -singleclickfns  execute functions immediately with one click
           -quit            quit
           -log <logfile>   write a log file
           -help            print this
```

A `.goof` file from PPM2OOF is loaded into OOF by typing `oof -grid filename.goof`. Compressed `.goof` files can also be read, (for instance, `oof -grid filename.goof.gz`) which speeds up reading as well as saving disk space.[1]

2.1 Example 1: Homogeneous Material Under Uniaxial Loading

This is a simple example that will hopefully give you a quick introduction to OOF and its general utilities. We will create a grid (finite element mesh) within OOF and perform a virtual uniaxial mechanical test on it.

[1] Be prepared to wait when reading in a large `.goof` file. A `.goof` file created from a 150×150 pixel map needs to allocate and store results for over 30 MBytes of memory.

1. **Start OOF:** Start up the OOF graphical interface: `oof`. This should create and display the main OOF menu, with the word "OOF" prominently in the upper left hand corner. (If this doesn't happen, then something is dreadfully wrong. Make sure that the compiled version of OOF matches your architecture. The graphical interface to OOF needs about 7 MBytes of available memory.)

2. **Initialize:** Launch the *Initialize Menu* by clicking on the word initialize in the Menus subwindow. The /initialize menu should occupy the real estate vacated by the main OOF menu.[2]

3. **Uniform:** You will see that there is currently only one type of grid — a uniform grid. Launch the /initialize/uniform menu by clicking on uniform in the /initialize menu.

4. **Mesh Parameters:** Set the height of the grid by clicking on height in the Variables list. A variable dialog box appear at the bottom right corner of the menu, with some explanatory text. You can type in the desired height in the dialog box; Let's make the sample taller than it is wide by setting the height equal to 1.5.[3] Hit the ⟨*return*⟩ key or click on the **Set** button to tell OOF that you're done changing the height variable. Notice that a new window pops up; this is the message box where information is stored and reported about the current state of OOF. Also notice that the new value of height appears in the Variable list. The values of nx and ny specify the number of triangular elements in the x-direction (width) and the y-direction (height). nx and ny of about 20 are about right if you want nearly instantaneous solutions, although you might try different values to see how they affect the mesh and the computation time.

5. **Function Window:** Click once on isotropic to launch a *Function Window* indicating that you want finite elements for a isotropic material to fill the grid. (If your X-Windows manager is putting a frame around the windows, then the words "initialize uniform isotropic" appear on the window frame. This is the path to this command from the *main OOF Menu*.)

6. **Set Material:** The *Function Window* for the isotropic element provides a table to fill out attributes for that particular type of element. The isotropic element has three thermoelastic parameters and the table allows you to specify the Young's modulus, the Poisson's ratio and the thermal expansion coefficient. *Note that if you hit a return in any of the function subwindows, it causes execution of that function. Use the mouse or the ⟨tab⟩ key to set more than one parameter.* Set the Young's modulus to 2 and the Poisson's ratio to 0.25. Let's imagine that we are doing the virtual experiment on a thin material and set the two-dimensional stress state to be plane stress by clicking

[2]A bug in the XForms library sometimes causes windows to move by the width of the window border when one Menu is replaced by another. There's nothing that can be done about this.

[3]The units in OOF are arbitrary—as long as you use consistent units, then your answers will (hopefully) make sense.

on the false button in the planestrain row. We won't be changing temperature in this example so the thermal expansion coefficient alpha does not matter. The value of gray controls how the mesh will be drawn; gray takes values between 0 and 1 inclusive, where 0 is black and 1 is white.

7. **Create the Mesh:** After all the information for the element has been entered, initialize the mesh by executing the function by clicking on the function name isotropic button at the top of the function window. Notice that the function execution is recorded in the message window. (Don't worry if you execute this function before you had intended, you can always recall the function and re-execute it.) The finite element mesh should now be ready.

8. **Graphics:** Look at the graphical representation of the mesh. Get to the main OOF Menu by either clicking on the **Home** button at the top of the /initialize/uniform submenu, or by backtracking by clicking on the **Back** button. From the main OOF Menu, open the graphics submenu. From the /Graphics Menu, open a *Drawer* by executing the open function by clicking on it twice quickly. (Clicking on it once would open a Function Window.) A Drawer should appear with a gray rectangle representing the isotropic material.

9. **Drawing Control:** Turn on the element edges so that the grid becomes apparent. The **Coordinates** Dashboard is currently visible. Switch to the **Attributes** Dashboard by clicking and holding the left mouse button on the *Dashboard Selector* and selecting **Attributes** from the pop-up menu. The controls visible in the Dashboard should change. Turn on the element edges by clicking on the button next to **Elements**. The mesh should be redrawn with edges turned on. The height, width and number of elements in the mesh were determined by the values you chose in the /initialize/uniform Menu.

10. **Pick Nodes:** Verify that the mesh has the height and width that you specified. Open the **Node Info** Dashboard on the Drawer. Move the mouse into the image region of the Drawer and place the hook of the "?" cursor over the top-right node and click with any button. (In this case, nodes are the vertices of the triangular elements.) You should see information appear in the Message Window reflecting the state of the Dashboard and the node which you queried. If you picked correctly, the node coordinates should be equal to width and height. The *node type* should be *xy* meaning that the node has an x- and a y- degree of freedom.

11. **BCs:** Set the boundary conditions. This is done by removing degrees of freedom from nodes. *Note tha this operation only affects the behavior of the equilibration stage of the solution. We will still be able to displace these nodes when distorting the mesh.* From the main OOF Menu, open the submenu bc. Fix the spatial degrees of freedom by picking the fix submenu. Let's fix both the x- and y- coordinates, by picking on the both submenu. The /bc/fix/both submenu will have names of *groups* of nodes

in the Function column. We will stretch the material in the vertical direction with fixed grips. Select top to remove the x- and y- degrees of freedom from the top row of nodes; do the same for the bottom nodes.

12. Set Distortion: Distort the mesh into the configuration for which you want a solution. From the main OOF Menu, open the distort submenu. Open the set submenu. There are a number of choices in the Variables column. Prepare to stretch the material by setting a ystrain of 0.20 (a strain of 20 percent in the vertical direction.) Now you have to specify which nodes you want to be distorted: click on the nodegroup top. Notice that in the graphics Drawer that nothing has happened; this is because the distortion has yet to be applied.[4]

13. Apply Distortion: Move **Back** one menu to /distort. Keep one eye on the graphics Drawer while you click on the increment function. You will see that the top elements have been distorted. To get the entire picture back into the Drawer either click the middle mouse on the **z**(oom) button at the top-left corner of the image subwindow of the Drawer or open the **Coordinates** dashboard and click on **Show Entire Image**.

14. Solution: From the main OOF Menu, click on the equilibrate function. Information about the solution will appear in the Message Window and the current graphical representation will appear in the image subwindow of the Drawer.

15. Stresses: Get a graphical picture of the stresses for the current solution. On the Drawer, hold down the left button on the top (wide) selector to bring up the list of Drawer types. Select **Stress invariant 1**. This will display the trace of the two-dimensional stress matrix: $\sigma_{xx} + \sigma_{yy}$.

16. Numerical Values: Change to the **Element Info** Dashboard on the Drawer. Turn on the **stress** button as well as several of the components of stress listed below it. Move the mouse onto the image and click on an element. The numerical values of the requested stress components in the element will appear in the message window.

17. Statistics: Switch the Drawer back to the **Mesh** display and open the **Attributes** Dashboard. We will be selecting elements so turn on the **Hot Elements** option, so that the selected elements are visible. Turn on the Click to Select… **Elements** option. In the image subwindow, drag a rectangle by holding down the mouse button as you move the mouse. The selected elements should change color. From the main OOF Menu, select the /output/stress/statistics submenu. Change the type variable to invariant (Click once on the word type in the Variables list. A pull-down menu of different statistics types appears at the bottom of the Menu. Use the left mouse button on the pull-down menu to see all the choices, or use the right or middle buttons

[4] By default, distortions are applied incrementally, so that, for example, you can step through a range of strains. The downside to this is that a distinction needs to be made between setting a distortion and applying it.

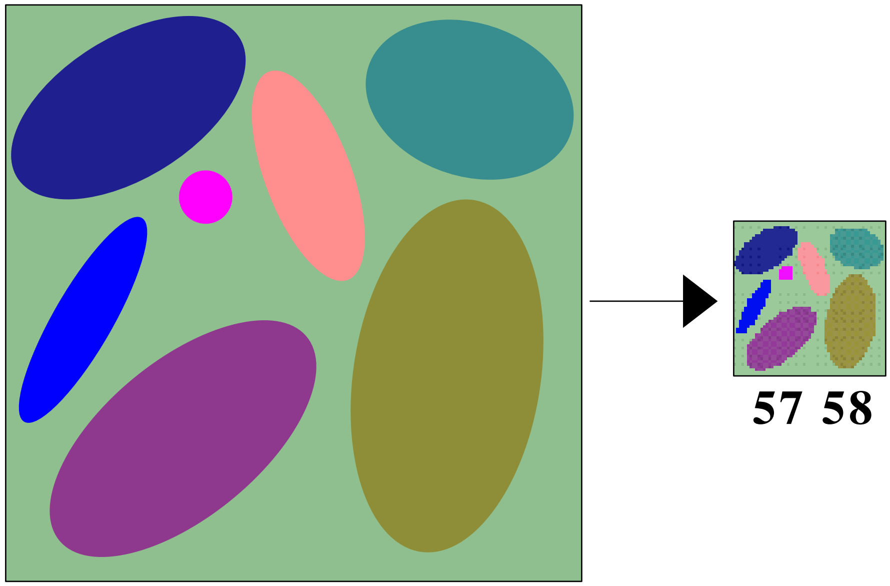

Figure 2.1: The image used to create data for example2.goof. The actual data was shrunk so that this particular example runs quickly.

---

to cycle through the choices.) Use the function selected to gather statistics on the selected elements. Output appears in the message window.

18. **Play, Quit:** You might want to play with various menus and see what's available. When you want to quit, go back to the main OOF Menu and click on the quit option in the Function list, or the **Quit** button at the top of the window.

2.2 Example 2: Using Data from PPM2OOF

This example illustrates how to use OOF with a `.goof` file (example2.goof, distributed with OOF) as data and gather statistics about the result. For demonstration purposes, example2.goof was created from an image created by a graphics "painting" program. The goof file contains definitions of *element groups*, which are simply lists of elements that can be acted on together by various commands. In this example, each group contains the elements making up different particles in a matrix, and the matrix itself.

1. **Read in the Data:** Start up the OOF graphical interface and read in the file example2.goof: `oof -grid example2.goof`.[5] This requires about 8 MBytes of memory,

[5] If the goof file has been compressed and has a .gz or .Z suffix, you do not have to uncompress it. Just invoke OOF with the full file name, including the suffix.

and should not take over a minute or so on most systems. You will need about 15 MBytes to do this entire example.

2. **Graphics:** Open a graphics drawer: /graphics/open. Bring up the **Attributes** Dashboard by choosing it from the Dashboard Selector (which is currently labeled "Coordinates"). On the **Attributes** Dashboard, turn **Edges** and **Hot Elements** on.

3. **Groups:** Observe the previously defined groups. From the main OOF Menu choose /select/elements/group. Notice that there are several group names in the **Functions** panel. Select any one of them and observe what happens on the image in the Drawer. The *current selection* is a special group which can be highlighted (as you should see), queried, or modified. Select another and notice that the new group is added to the current selection. To isolate another group, you have reset the current selection. Click on the **Freeze** button on the top of the Menu (this will cause the current menu to remain in place as you call up other menus). Click on the **Back** button, you should get an additional menu: /select/elements. Launch the none function window by double-clicking. Close the new Menu you created to avoid crowding by clicking on the **Close** button. Click on the **Freeze** button of the none function window you just created and then execute by clicking on the word "none". Select some of the other groups one at a time.

4. **Query:** Open the **Element Info** Dashboard on the Graphics Drawer and indicate that you want **element type** and **element parameters**. Move the mouse into the image and click on some elements in the various groups and observe what types of elements are in each of the groups. The elements in each of the particles should have orthorhombic thermoelastic properties, but each group should be in a different orientation. A single entry in a orientation parameter indicates a counterclockwise rotation around the z-axis. Notice the orientation for an element in the "doors" group. It has a triple entry for the orientation, indicating a full three dimensional rotation via Euler angles (see Section 4.3 for the details).

5. **Thermal Stresses:** From the main Menu, increase the uniform temperature field by setting the deltaT variable to 0.1. Execute the equilibrate function. The sample should expand—even though the matrix (group 'stage') has an isotropic thermal expansion coefficient of zero—because the particles have positive expansion coefficients. If you want to display the entire image, click middle mouse button on the zoom button **z** in the Drawer. Turn off the **Edges** in the **Attributes** Dashboard. Display the **Stress Invariant 1** field by choosing it in the **Drawing Selector**. (It may take a little while for the stresses to be calculated and displayed). Notice that the particles tend to have a negative (compressive) component to their hydrostatic stresses.

6. **Numerical A:** Get statistics about the stresses in the various particles. In the /output/stress/statistics menu, set type to invariant. Click on some of the group names in the **Functions** menu, and observe the output; the averages and standard deviation

of the largest and smallest eigenstresses are displayed, as well as the maximum value of shear (which is just half the difference in the eigenvalues). Turn the type back to cartesian and obtain the statistics for the stresses in the screen coordinates.

7. **Numerical B:** Find the average stresses in all the particles by creating a supergroup. Go back to the **Mesh** Drawer and make sure that no elements are currently selected (/select/elements/none). Select all the particles from the /select/elements/group Submenu.[6] *I.e*, click once on all the groups except stage in the **Functions** panel. Create a new group. From the /groups/elements Submenu, double click on the new function and enter the new group name, `particles`, in the Function window, and then hit ⟨*return*⟩ or click on the **new** button to execute the function. From the /groups/elements/add_elements menu, execute the function particles to add the current selection to this new group. Now, from the /output/stress/statistics Submenu, interrogate the statistics of this new group which should now appear in the **Functions** subwindow.

8. **Graphics:** Return to the **Stress invariant 1** display on the drawer. In the **Color** Dashboard, change the **size** of the Color Map to about 8 (type 8 and hit ⟨*return*⟩ and observe how choosing *less* information can sometimes give you a much clear picture of how the stresses are distributed. Display only those parts of the microstructure which are in compression: Turn off the **Full Scale** option on the Dashboard and then set the upper limit of the **Scale** to 0 so that only negative values have a color. It may be instructive to **Flip** the color scale.

9. **Reset and Retest:** Run a new virtual experiment on this microstructure without restarting the program. From the main OOF Menu, set deltaT to a new value (zero is a good choice) and execute the function reset. (The image will be drawn with stresses that do not correspond to equilibrium, the **Mesh** Drawer will make more sense.) With the bc Submenu, fix/both degrees of freedom on the top and bottom node groups. With the distort Submenu, set xshear to `0.1` for the top node group. Don't forget to increment the distortion in the distort Submenu. Find the solution by executing equilibrate from the main OOF Menu and investigate the solution.

10. **Modify Grid:** See if you can modify the current grid by selecting—either by group or with the mouse in the **Mesh** Drawer—a set of elements and use the /modify/replace/ Submenu and the empty function to remove elements.

[6] This window should still be up, unless you weren't following directions.

Chapter 3

The Menus

A *menu*[1] organizes or catalogs particular utilities onto one page. Other menus (*submenus*) can be called from a menu. Each submenu represents a more specific catalog. The title of the menu is displayed in the upper left corner along with (if it is a submenu) a few words describing the utility of the current menu. Each of the menus has the same form. It contains a left panel of **Submenus**, a middle panel of **Functions**, and a right panel of **Variables**. These columns represent most of the uses of OOF and will be described in this chapter.

There are five buttons on top of the menu. From left to right the buttons are:

Home Return to the very top menu (the main OOF Menu).

Back Return to the next highest menu (on the main OOF Menu, this button is labeled **Quit** and pushing it has a not unpredictable consequence).

Close Remove the Menu from the display if there are other menus currently being displayed. If no other menus are being displayed, in a Submenu **Close** has the same effect as the button **Home** and no effect in the main OOF Menu.

Freeze Make the window permanent. If a Submenu is selected, it appears as new display. Multiple displayed menus are very useful for executing functions consecutively which don't appear on the same Submenu.

Record Record a sequence of commands to create a *Macro Function* in the current Menu. For instance, from the main OOF Menu, press **Record** and then bc, fix, both, top, and bottom. Then return **Home** and turn off **Record** (now labeled **Stop**). You will be queried for a name to represent this sequence of commands (e.g., `grip_top`) and that name will appear as a new Function on the main OOF Menu.

Finally, at the bottom of the menu is a region for setting the values of variables.

[1] This section briefly summarizes the information in Section 1.5.

3.0.1 The OOF Tree

OOF is organized as a tree of submenus. The following list describes that tree structure as well as the section where that Submenu and its Functions and Variables are described.

- initialize (3.2)
  - uniform (3.13)
- modify (3.3)
  - replace (3.14)
  - select (3.15)
    - type (3.15)
    - group (3.15)
- groups (3.4)
  - elements (3.16)
    - show (3.16)
    - delete (3.16)
    - add_elements (3.16)
    - remove_elements (3.16)
    - select (3.16)
      - type (3.16)
      - group (3.16)
    - unselect (3.16)
  - nodes (3.17)
    - delete (3.17)
    - add_nodes (3.17)
    - remove_nodes (3.17)
    - select (3.17)
    - unselect (3.17)
- select (3.5)
  - nodes (3.18)
  - elements (3.19)
    - type (3.20)
    - group (3.21)
- bc (3.6)
  - fix (3.22)
    - x (3.22)
    - y (3.22)
    - both (3.22)
  - free (3.23)
    - x (3.23)
    - y (3.23)
    - both (3.23)
  - enslave (3.24)
    - x (3.24)
    - y (3.24)
    - both (3.24)
  - emancipate (3.25)
    - x (3.25)
    - y (3.25)
    - both (3.25)
  - groups (3.17)
    - delete (3.17)
    - add_nodes (3.17)
    - remove_nodes (3.17)
    - select (3.17)
    - unselect (3.17)
- distort (3.7)
  - set (3.27)
  - clear (3.28)
  - show (3.29)
- output (3.8)
  - tee (3.30)
  - grid (3.31)
    - matrix (3.31)
  - individual (3.32)
    - stress (3.33)
    - stress (3.33)

  - force (3.35)
  - displacement (3.36)
  - stress (3.37)
    * statistics (3.38)
  - strain (3.39)
  - cross_section (3.40)
    * uniform (3.40.1)
    * nonuniform (3.40.2)
- graphics (3.9 and 5)
- plot (3.10)
- macros (3.11)
- log (3.12)

3.1 The Main OOF Menu

The main OOF Menu has the following composition:

Menus	Functions	Variables
initialize	commandfile	deltaT = 0
modify	equilibrate	maxiter = 1000
groups	comp_equil	tolerance = 1e-05
select	mutate	globalK = true
bc	reset	preconditioner = ILU
distort	cruise	verbose = true
graphics	loop	seed = 17
plot	quit	
macros		
log		

The main Submenus are:

initialize Initialize a grid from scratch or read from a file. (Section 3.2.)

modify Modify elements in existing grid. (Section 3.3.)

groups Manage node and element groups. (Section 3.4.)

select Select nodes and elements and modify current selection. (Section 3.5.)

bc Apply boundary conditions to nodegroups. (Section 3.6.)

distort Distort mesh by displacing nodegroups. (Section 3.7.)

output Print various types of numerical information. (Section 3.8.)

graphics Manage graphics Drawers. (Section 3.9 and Chapter 5.)

plot Plot information into a file. (currently in development). (Section 3.10.)

macros Macro management: save and read macros. (Section 3.11.)

log Keep a session logfile. (Section 3.12.)

The functions and variables for each menu are described in separate sections. The functions and variables are for the main OOF Menu are described in this section.

3.1.1 /commandfile

Read in commands from a text file. The name of the file can be entered on the commandfile Function Window.

For example, reading a file which has the contents

```
bc fix both top
bc fix both bottom
distort set ystrain = 0.1
distort set top
distort increment
equilibrate
```

will set boundary conditions and distortions and then equilibrate. See Section 6.1 for information on OOF's text interface. The /log menu can be used to create commandfiles (Section 3.12).

commandfile arguments

filename The name of the file from which commands are read. Default: inputfile.

3.1.2 /equilibrate

Find the solution for the current boundary conditions, distortion and temperature.

The solver uses an iterative sparse matrix method.[2] The number of iterations which the solver will attempt is fixed by the variable **maxiter**. If convergence to a solution is not achieved by **maxiter** iterations, then equilibrate can be re-invoked to start from the current state.

When solving the matrix equation $\mathbf{Kx} = \mathbf{f}$, the solution is considered to be converged when the norm of the residual $\mathbf{Kx} - \mathbf{f}$ is less than the variable **tolerance** times the norm of $\mathbf{f}$. $\mathbf{f}$ is the vector of external and thermal forces, $\mathbf{K}$ is the stiffness matrix, and the vector $\mathbf{x}$ is the degrees of freedom being solved for.

The solution method can be adjusted by the variable **preconditioner** and the boolean variable **globalK**. The preconditioner is a matrix M which multiplies both sides of $\mathbf{Kx} = \mathbf{f}$. A good preconditioner nearly diagonalizes $\mathbf{K}$ and hence speeds up the iterative solution. The iterative method does not actually require that the matrix $\mathbf{K}$ be constructed and stored as a global matrix—only that its individual components can be computed when needed. However, some of the preconditioners require the global matrix. Set variable **globalK** to `true` if you want the global matrix to be computed. The choices for preconditioners are

`none` Use no preconditioner ($\mathbf{M} = \mathbf{1}$).

[2] In particular, it uses the Conjugate Gradient routine from the IML++ library, `http://math.nist.gov/iml++/`.

`diagonal` $\mathbf{M}$ is the inverse of the diagonal elements of $\mathbf{K}$. This works well only if $\mathbf{K}$ is strongly diagonally dominant, but the preconditioning step is fast and simple.

`block` This is the same as diagonal, but $\mathbf{M}$ is the inverse of the block diagonal elements of $\mathbf{K}$, where each block corresponds to the x and y degrees of freedom of one node in the mesh. This doesn't work as well as we'd hoped, but it's still fast to implement.

`ILU` Incomplete LU decomposition. This seems to work the best, but requires globalK = `true`.

`ICP` Incomplete Cholesky decomposition. Also requires globalK = `true`.

3.1.3 /comp_equil

Perform a complete equilibration. Elements can be *mutable*, changing their properties in response to their current state (see Chapter 4). In a *complete equilibration* the system is equilibrated and elements are mutated in response to their new configuration. If any elements mutate, the system is re-equilibrated and mutated again, until no more elements mutate. The equilibration is affected by the same variables as the main OOF Function equilibrate.

Pressing the big red STOP button that pops up will interrupt the cycle of mutation and equilibration after the next set of mutations is completed. You may need to explicitly equilibrate afterwards.

3.1.4 /mutate

Check current solution state to see if any mutable elements should change, and change them. To mutate and re-equilibrate repeatedly until no more elements mutate, use comp_equil (Section 3.1.3).

3.1.5 /reset

Resets all the degrees of freedom to have zero displacements. It zeroes the absolute distortions but does *not* change the incremental distortions. (See Section 3.27 for a discussion of incremental and absolute distortions.) It also does *not* affect the boundary conditions. (Use /bc/free/all (Section 3.23.1) to remove boundary conditions.)

3.1.6 /cruise

Start up an increment–equilibrate loop which runs for a user-specified number of iterations. (See Section 3.7 for a description of increment.) A window with a big red **Stop** button will appear. Click on the button to abort the loop.

cruise arguments

iterations The number of iterations in the cruise loop. Default: 0.

3.1.7 /loop

Starts up a user-defined loop with 'initialize,' 'body,' and 'finish' components. The loop runs for a user-specified number of iterations. The commands can be any typable OOF commands. See Section 6.1. When the loop starts, a window with a big red **Stop** button will appear. Click on the button to abort the loop.

loop arguments

initialize A statement to executed once before the loop starts. Default: "" (an empty string, in quotation marks. The quotation marks are optional if you're using the graphics interface).

body A statement to be executed repeatedly for a user-specified number of iterations. Default: "".

finish A statement to executed after the loop has been completed. Default: "".

iterations The number of times that the body statement will be executed. Default: 0.

Example: The loop:

```
initialize = "deltaT = 0.05"
body = "equilibrate ; mutate"
finish = "graphics open"
iterate = 3
```

will set the temperature, equilibrate and mutate three times and then open a graphics window.

3.1.8 /quit

Quits after confirmation. A dialog box pops up asking "Save session log before quitting?" Clicking on "Save" or hitting return brings up another file browser for the log file. The log file contains all the commands issued since the start of the session. To save a subset of commands, see /log/start and /log/stop (Section 3.12. To recreate the session, load the log file with the `-file` command line option (Section 1.4), or the /commandfile function (Section 3.1.1).

If there's nothing to log, OOF won't offer to let you save the log file.

3.1.9 Main OOF Menu variables

deltaT The value of the uniform superheating and the term which is multiplied by the thermal expansion coefficients to get a thermal strain. A deltaT of zero is the stress-free strain state. Default: 0.

maxiter The maximum number of iterations which will be performed by a single equilibration step. Control is returned to the menu when the number of iterations exceeds **maxiter**, whether convergence has been attained or not. See Section 3.1.2. Default: 1000.

tolerance The tolerance of the iterative method used to find the solution. See Section 3.1.2 for details. Default: 1.0e–05.

globalK A boolean (true/false) variable indicating whether a global stiffness matrix should be calculated (speeding up calculation, but taking up memory) or not. See Section 3.1.2 for details. Default: true.

preconditioner Set the preconditioner used during equilibration. Section 3.1.2 discusses what the preconditioner does and what the options are. Default: ILU.

seed The initializer for the random number generator used in various places (*e.g*, /modify/select/swisscheese.) default: 17.

3.2 /initialize

This menu creates a finite element grid, either by constructing it from scratch or reading it from a file.

Menus	**Functions**	**Variables**
uniform	file	
	show_types	

The Submenus are:

uniform Initialize a uniform homogeneous mesh. See section 3.13

3.2.1 /initialize/file

Read a `.goof` (a grid file) from a user-specified filename. The file can be one saved by OOF (see Section 3.31), or one written by PPM2OOF.

file arguments

filename The name of the file to be read. Default: grid.goof.

3.2.2 /initialize/show_types

Lists (in the Message Window) the types of elements and nodes that the current version of OOF knows about.

3.3 /modify

This menu allows the user to replace or modify existing elements.

Menus	Functions	Variables
replace		
select		

The Submenus are:

replace Replace the currently selected elements. See section 3.14

select Methods for selecting and unselecting elements. See section 3.19.

3.4 /groups

This menu allows the user to manage node and element groups. Many operations in OOF act upon groups of nodes or elements, or upon the currently *selected* nodes or elements. For example, you can query the average stress in a group of elements, or apply a distortion to a group of nodes. One way of making a selection is to select a predefined group. One way of defining a group is to give a name to the current selection.

Menus	Functions	Variables
elements		
nodes		

The Submenus are:

elements Manage the element groups. See section 3.16

nodes Manage the node groups. See section 3.17

3.5 /select

Methods for selecting and unselecting elements and nodes. (Elements and nodes can also be selected graphically. See Section 5.2.2.)

Menus	Functions	Variables
nodes		
elements		

The Submenus are:

nodes Select the nodes by nodegroups. See section 3.18

elements Methods for selecting elements. See section 3.15

3.6 /bc

This Submenu has methods for setting boundary conditions, or removing degrees of freedom from the nodes.

In OOF, *boundary nodes* are nodes whose positions are fixed and therefore aren't solved for by the equilibrate command. Actually, the x and y degrees of freedom are treated separately, so a node can be a boundary node for x and not for y, and vice versa. The /bc menu determines which degrees of freedom are on the boundary, but does not determine the positions of those degrees of freedom. The /distort menu sets the positions of the boundary degrees of freedom.

The bc menu also is used to *enslave* degrees of freedom to one another. Enslaved degrees of freedom all have the same displacement. In other words, they're all actually just one single degree of freedom. Enslaved degrees of freedom may also be fixed (*ie,* on the boundary), in which case they aren't very interesting, because their positions won't change during equilibration, and the fact that they're enslaved is irrelevant. Enslaved degrees of freedom which aren't fixed, however, change their positions in unison during equilibration. For example, to simulate a system below a rigid plate with a weight on it, you'd enslave the y degrees of freedom on the top edge of the sample and apply a y force to it.

Menus	Functions	Variables
fix	show	
free		
enslave		
emancipate		
groups		

The Submenus are:

fix Fix a degree of freedom. See section 3.22

free Remove boundary conditions at nodes. See section 3.23.

enslave Enslave the nodes in a group. See section 3.24.

emancipate Emancipate (*ie,* un-enslave) the nodes in a group. See section 3.25.

groups Manage node groups (same as Submenu /groups/nodes). See section 3.17.

3.6.1 /bc/show

Show (in the message subwindow) the current active degrees of freedom for each of the nodegroups.

3.7 /distort

Methods for setting and incrementing the mesh distortion.

This menu sets the positions of the boundary degrees of freedom. See the comment in Section 3.6 for the distinction between setting boundary conditions and setting distortions.

Two kinds of distortions can be set in OOF: absolute and incremental. Setting an *absolute* distortion sets the actual position of or force on the nodes. Setting an *incremental* distortion sets an amount to be added to the position or force at a later time, when the /distort/increment function is executed.

Menus	Functions	Variables
set	increment	
show	stealth_increment	

The Submenus are:

set Set and control the current distortions prior to increment. See section 3.27

show Show (in the message subwindow) information about the current distortion state. See section 3.29

3.7.1 /distort/increment

Increment the current distortion of the mesh by the incremental distortion of each node group.

3.7.2 /distort/stealth_increment

Just like /distort/increment, but does not cause the Graphics Drawers to update. For very large meshes, a significant amount of time can be taken for a redraw command. This function allows you to skip the non-equilibrium redraw.

3.8 /output

Methods for getting ASCII numerical information.

Menus	Functions	Variables
tee grid	node energy	
individual		
force		
displacement		
stress		
strain		

The Submenus are:

tee Send output to a file as well as to the screen. See section 3.30.

grid Save the grid as a `.goof` file. See section 3.31.

individual Print various values for each element individually. See section 3.32.

force Print the forces for nodegroups. See section 3.35.

displacement Print the displacements for nodegroups. See section 3.36.

stress Get stress information about element groups. See section 3.37.

strain Get strain information about element groups. See section 3.39.

cross_section Plot a function along a given line in the xy plane. See section 3.40.

3.8.1 /output/node

Print information about a particular node in the Message Window. This can also be done in the Drawers without having to know the node number, but if you want to repeatedly query a single node, this function is more convenient. Use the **Node Info** Dashboard in any Drawer to find node indices.

node arguments

node_number The index of the node in question. Default: 0.

3.8.2 /output/energy

Print the current total elastic energy in the message subwindow.

3.9 /graphics

Open and close Graphics Drawers (see Chapter 5).

Menus	Functions	Variables
	open	wallpaper
	close_all	enhancement

3.9.1 /graphics/open

Open a new graphics drawer.

3.9.2 /graphics/close_all

Close all graphics drawers.

3.9.3 /graphics variables

wallpaper A boolean variable that determines whether the background for images in the Drawers is white (wallpaper=`false`) or patterned (wallpaper=`true`). It can be easier to see lightly colored elements against a patterned background.

enhancement A floating point number describing how the displacements of the nodes in the mesh should be enhanced when drawn. The nodes are drawn at their original positions plus enhancement times their displacements. With enhancement=0, nodes are drawn at their original undistorted positions, with enhancement=1 they have their actual displacements, and with enhancement>1 the displacements are exaggerated. Default: 1.0

3.10 /plot

This menu is currently in development. There is some functionality here, but it will be changing soon.

3.11 /macros

Load and save macro definitions. Macros are short-hand names for a sequence of commands which are built up using the **Record** button or written into a file. See the discussion in Section 1.5.1 for more detail. This menu allows the user to save defined macros, or read them from a file.

Menus	Functions	Variables
	save	
	load	

3.11.1 /macros/save

Save all the macros to a user-specified filename.

save arguments

filename A string indicating the name of the file. Default: macrofile.

3.11.2 /macros/load

Load all the macros from a user-specified filename. Macro files can also be read by /commandfile (Section 3.1.1). The only difference is whether or not the macros are echoed to the screen as they're read.

load arguments

filename A string indicating the name of the file. Default: macrofile.

3.12 /log

Using this menu, you can save a sequence of commands into a file which you may subsequently edit and re-run using the /commandfile function. This is a useful way of getting OOF automatically back to some state that you want to investigate.

The Message Window (Section 1.5.2) contains both output and commands. It can be saved separately, but cannot be loaded. The log file contains only commands.

Menus	Functions	Variables
	start	
	stop	
	save	

3.12.1 /log/start

Starts saving the commands in a user-specified command file.

start arguments

file A string indicating the name of the log file. Default: cmdfile.oof.

3.12.2 /log/stop

Stop saving commands.

3.12.3 /log/save

Save *all* the commands executed since the start of the program, not just those executed since executing /log/start. This comes in handy when you've forgotten to use /log/start.

save arguments

file A string indicating the name of the log file.

3.13 /initialize/uniform

This menu creates a uniform rectangular grid (finite element mesh) of a single type of element. Variables in the menu determine the shape of the rectangle, how it is divided into finite elements, and what type of nodes to use. The grid is then initialized by executing a function representing one of the element types. After a uniform grid is created, the grid can be made inhomogeneous by selecting elements (section 3.19) and replacing them (section 3.3).

The size of the grid is determined by the variables height and width. The grid is divided into nx × ny rectangular elements, each of which is then divided into two triangular elements according to the variable diagonals.

Menus	Functions	Variables
	empty	nx = 20
	isotropic	ny = 20
	cubic	width = 1
	hexagonal	height = 1
	orthorhombic	diagonals = liberal
	eds_el	interior_nodes = xy
	damisotropic	boundary_nodes = xy
	damage	

3.13.1 /initialize/uniform/isotropic, *etc.*

The entries in the Function column list the possible types of elements which will homogeneously fill the uniform grid. All elements are described in chapter 4.

3.13.2 /initialize/uniform variables

nx The number of element edges to create in the x (horizontal) direction. Default: 20.

ny The number of element edges to create in the y (vertical) direction. Default: 20.

width The width of the rectangle in the same system of units that elastic parameters will be specified. The thickness (out-of-screen) physical dimension is assumed to be one, so all results (or scale moduli appropriately) should be interpreted with this in mind. Default: 1.

See section 4.4 for more discussion about units.

height The height of the rectangle in physical units. Default: 1.

diagonals A choice variable for how to draw the diagonals in each rectangle. Choices are: `liberal` (all diagonals point to the left); `conservative` (all diagonals point to the right); `moderate` (half liberal, half conservative); and `anarchic` (randomly assigned diagonals); Default: liberal

interior_nodes A choice variable for which types of degrees of freedom are allowed at the interior nodes. Choices: 1) `xy` (both degrees of freedom) 2) `linear` (allow movement along a general line only. We once thought this was useful but have forgotten why.

The linear node maintains its own coordinate system. Initially the nodes' x and y axes are lined up with the screen axes, but when a shear distortion is applied the axes rotate (Section 3.27). The boundary conditions are applied in the node's coordinate system (Section 3.6). For example, if the nodes on the bottom edge of the grid are linear nodes, a `yshear` distortion has been applied, and the nodes have been fixed in y, then during equilibration they can move only along the line $y = x \times \text{yshear}$.) Default: xy

boundary_nodes A choice variable for which types of degrees of freedom are allowed at the boundary (edges of the rectangle) nodes. Same choices as interior_nodes. Default: xy

3.14 /modify/replace

Replaces the current element selection with elements chosen from the types in the Function column.

Menus	Functions	Variables
	empty	
	isotropic	
	cubic	
	hexagonal	
	orthorhombic	
	eds_el	
	damisotropic	
	damage	

3.14.1 /modify/replace/isotropic, *etc.*

The entries in the Function column list the possible types of elements which will replace the current selection. All elements are described in chapter 4.

3.15 /modify/select

This menu is identical to the /select/elements Submenu. It is repeated here for convenience. See Section 3.19.

3.16 /groups/elements

This menu manages and modifies element groups.

Menus	Functions	Variables
show	new	
delete		
add_elements		
remove_elements		
select		
unselect		

The Submenus are:

show A menu of the current element groups. Clicking on a group prints the number of elements in the group.

delete A menu of the current element groups. Clicking on a group deletes the group. The elements themselves aren't touched, but the group is disbanded.

add_elements A menu of the current element groups. Clicking on a group adds the currently selected elements to the chosen group. Elements can be selected with the /select/elements Menu (Section 3.19) or its equivalents (/modify/select and /groups/elements/select) or with the mouse in the **Mesh Drawer** (Section 5.2.2).

remove_elements A menu of the current element groups. Clicking on a group removes the current selection from a chosen group. Actually it only removes the intersection of the current selection and the chosen group.

select The menu for selecting elements. This is the same menu as /select/elements, repeated here for convenience. See section 3.19.

unselect A menu of the current element groups. Clicking on a group unselects any members of a group which are in the current selection.

3.16.1 /groups/elements/new

Define a new group with a user-specified name. The new group will contain no elements. After creation, elements can be added through /groups/elements/add_elements.

new arguments

name A string identifying the name of the new group. After creation, this group will appear in all menus containing lists of groups. The string should not contain spaces. Default: groupname.

3.17 /groups/nodes

Manage and modify nodegroups. Eight nodegroups are always created in any grid.[3] There are four groups of nodes along the edges of the rectangle (right, left, top, bottom) and four groups each containing a single node at the corners (upperleft, lowerleft, upperright, lowerright). The node in the upperleft group is also in the top and left nodegroups.

Menus	Functions	Variables
delete	new	
add_nodes		
remove_nodes		
select		
unselect		

The Submenus are:

delete A menu of the current nodegroups. Each function deletes a group of nodes. It doesn't delete the nodes, just the group.

add_nodes A menu of the current nodegroups. Each function adds the current selection to the chosen group.

remove_nodes A menu of the current nodegroups. Each function removes the current selection from the chosen group.

select Select and unselect nodes. This menu is the same as /select/nodes (Section 3.18) and is repeated here for convenience.

unselect A menu of the current nodegroups plus an additional all function. /groups/nodes/unselect/all has the same effect as /groups/nodes/select/none.

3.17.1 /groups/nodes/new

Define a new group with a user-specified name. The new group will contain no nodes. After creation, elements can be added through /groups/elements/add_addnodes.

new arguments

name A string identifying the name of the new group. After creation, this group will appear in all menus containing lists of groups. The string should not contain spaces. Default: groupname.

[3]This isn't a fundamental statement—grids created by the /initialize/uniform commands in OOF or by PPM2OOF have these nodegroups, but a `.goof` file created by some other program need not have them.

3.18 /select/nodes

Select nodes. Nodes may also be selected by using the mouse in the **Mesh Drawer** (see Section 5.2.2).

Menus	Functions	Variables
	all	
	number	
	toggle	
	none	
	groupname...	
	⋮	

3.18.1 /select/nodes/all

Select all the nodes in the grid.

3.18.2 /select/nodes/number

Select the node with the given node number. You can find out a node's number with the "node index" button in the **Node Info** Dashboard.

3.18.3 /select/nodes/toggle

Select all the nodes which are currently unselected, while unselecting all the currently selected nodes.

3.18.4 /select/nodes/none

Unselect all the nodes in the grid.

3.18.5 /select/nodes/*groupname*

All of the currently defined nodegroups appear in the menu. Clicking on a group name selects all the nodes in the group.

3.19 /select/elements

This menu contains various tools for selecting elements. Elements may also be selected by using the mouse in the **Mesh Drawer** (see Section 5.2.2).

Menus	Functions	Variables
type	all	
group	toggle	
	none	
	number rectangle	
	circle	
	random	
	swisscheese	
	gray	

The submenus are:

type Select elements by element type or characteristics. See section 3.20.

group Select a previously defined element group. See section 3.21)

3.19.1 /select/elements/all

Select all the elements.

3.19.2 /select/elements/toggle

Select the complement to the current selection. In other words, select all unselected elements and unselect all selected elements.

3.19.3 /select/elements/none

Unselect all elements.

3.19.4 /select/elements/number

Select the element whose index is the variable **n**. This is useful in tandem with the commands in the /output/individual menu. See section 3.32.

3.19.5 /select/elements/rectangle

Select all elements inside a user-specified rectangle and add them to the current selection. A triangular element is included in the selection if its center point is inside the specified rectangle.

rectangle arguments

All arguments are in physical units (*i.e.* the same as width and height).

x0 The lower left hand x-coordinate of the selection rectangle. Default: 0.

y0 The lower left hand y-coordinate of the selection rectangle. Default: 0.

x1 The upper right hand x-coordinate of the selection rectangle. Default: 0.

y1 The upper right hand y-coordinate of the selection rectangle. Default: 0.

3.19.6 /select/elements/circle

Select all elements inside a user-specified circle and add them to the current selection.

circle arguments

All arguments are in physical units (*i.e.* the same as width and height).

x The x-coordinate of the circle's center. Default: 0.

y The y-coordinate of circle's center. Default: 0.

radius The circle's radius. Default: 0.

3.19.7 /select/elements/random

Select elements at random. The previously selected elements are unselected first. The seed for the random number generator is the variable **seed** in the main OOF menu.

random arguments

probability The probability that an element will be selected. The fraction of elements selected will be near the probability value for a large number of elements.

3.19.8 /select/elements/swisscheese

Select non overlapping circles of elements. Sometimes a particular set of swisscheese parameters will make proper selection time-consuming, if not impossible. The function does a quick job and then leaves details about how well it performed in the message subwindow.

swisscheese arguments

Spatial arguments are in physical units (*i.e.* the same as width and height).

ncircles The number of circles which OOF will attempt to select. The actual number may fall short. Default: 1.

Ravg The average radius of the circles. Default: 0.

Rdev The deviation about the average radius. The actual radius is $Ravg + pRdev$, where p is a random number chosen from a Gaussian distribution with mean 0 and deviation 1.

separation The minimum distance separating the edges any two circles. Default: 0.

maxtries The number of times to try to select a circle if selection becomes difficult. No attempt is made to place the circles intelligently. A circle is placed at random, and if it overlaps or comes within separation of another circle, it is rejected. After it is rejected maxtries times, the algorithm gives up and goes on to the next circle. Default: 100.

3.19.9 /select/elements/gray

Select elements by with gray values in a user-specified range. Gray takes values between 0 and 1.

gray arguments

minimum The minimum (darker) gray value to select. Default: 0.

maximum The maximum (lighter) gray value to select. Default: 1.

3.20 /select/elements/type

This menu allows the user to select elements by type.

Menus	Functions	Variables
	all	
	none	
	interior	
	boundary	
	nonempty	
	isotropic	
	etc...	

3.20.1 /select/elements/all

Select all the elements in the grid.

3.20.2 /select/elements/none

Unselect all the elements in the grid.

3.20.3 /select/elements/interior

Select all elements containing no fixed nodes.

3.20.4 /select/elements/boundary

Select all elements containing at least one fixed node.

3.20.5 /select/elements/nonempty

Select all elements that aren't empty elements. See Section 4.5.2.

3.20.6 /select/elements/isotropic, *etc.*

Select elements of the given type. See Section 4 for a description of the element types.

3.21 /select/elements/group

The names of the functions in this menu are the names of the element groups which have been defined. (See Section 3.16.) Executing one of the functions selects all the elements in the named group.

3.22 /bc/fix

This menu allows the user to fix displacements (remove degrees of freedom from) node-groups. Fixed degrees of freedom can still be moved via applied distortions (Section 3.7) but do not move when the system is equilibrated.

Menus	Functions	Variables
x		
y		
both		

The Submenus are:

x A menu of the current nodegroups which freezes the x degree of freedom for the entire nodegroup.

y A menu of the current nodegroups which freezes the y degree of freedom for the entire nodegroup.

both A menu of the current nodegroups which freezes both the x and y degrees of freedom for the entire nodegroup.

3.23 /bc/free

This menu allows the user to remove constraints from nodegroups, undoing the effect of /bc/fix. Since a node and its degrees of freedom can be in more than one nodegroup, degrees of freedom are not actually free until they've been freed in all nodegroups to which they belong.

Menus	Functions	Variables
x	all	
y		
both		

The Submenus are:

x A menu of the current nodegroups which frees the x degree of freedom for the entire nodegroup.

y A menu of the current nodegroups which frees the y degree of freedom for the entire nodegroup.

both A menu of the current nodegroups which frees both degrees of freedom for the entire nodegroup.

3.23.1 /bc/free/all

Removes all boundary conditions, freeing all degrees of freedom.

3.24 /bc/enslave

This menu allows the user to require that the x or y displacements of all nodes in a nodegroup be identical. The x or y degree of freedom for all nodes in a nodegroup is replaced by a single master degree of freedom. The original degrees of freedom are said to be *enslaved* to the master.

The displacement of the slaves is the same as the displacement of the master. The displacement of the master is initially set to the average displacement of the slaves. If any of the slaves are fixed (see Section 3.22), so is the master. All forces applied to the slaves are applied to the master (see Section 3.27). Note that when querying the force on an enslaved degree of freedom with the **Node Info** Dashboard (Section 5.2.5) the force on the *master* is reported, which is the sum of the forces on all (x or y) degrees of freedom in the group.

Menus	Functions	Variables
x		
y		
both		

The Submenus are:

x A menu of the current nodegroups which enslaves the x degree of freedom for the entire nodegroup.

y A menu of the current nodegroups which enslaves the y degree of freedom for the entire nodegroup.

both A menu of the current nodegroups which enslaves both the x and y degrees of freedom for the entire nodegroup.

3.25 /bc/emancipate

This menu undoes the effect of the commands in /bc/enslave. When degrees of freedom in a nodegroup are emancipated, their master is removed and each degree of freedom can have a different displacement.

Menus	Functions	Variables
x		
y		
both		

The Submenus are:

x A menu of the current nodegroups which emancipates the x degree of freedom for the entire nodegroup.

y A menu of the current nodegroups which emancipates the y degree of freedom for the entire nodegroup.

both A menu of the current nodegroups which emancipates both the x and y degrees of freedom for the entire nodegroup.

3.26 /bc/groups

This menu is the same as /groups/nodes and is described in section 3.17.

3.27 /distort/set

This menu allows the user to specify the displacements of or forces upon the degrees of freedom of the nodegroups. It is meaningless to specify displacements for nodes that haven't been fixed by the /bc/fix commands, since those nodes will move when the system is equilibrated.

The *current distortion* is made up of all the distortions and forces in the **Variables** panel.

There are two steps involved in setting the a distortion for a group of nodes. First, the current distortion is set by adjusting the variables in this Menu. Second, the current distortion is applied to a particular set or sets of nodes by selecting nodegroups from the **Functions** panel.

Finally, there are two different types of distortion: *incremental* and *absolute* (or not incremental). This menu allows you to set each of them independently, depending on the state of the boolean variable incremental. The /distort/increment function needs to be invoked to make the nodes move according to the incremental distortions which have been set. If incremental is false, then the absolute distortions are being set and the displacements take place immediately. In other words, setting an absolute distortion sets the actual position and force on the nodes, whereas setting an incremental distortion sets only the amount by which the position and force will change when the increment function executed.

Menus	**Functions**	**Variables**
	reset_params	xstrain = 0
	all_nodes	ystrain = 0
	boundaries	xshift = 0
	nodegroup...	yshift = 0
	⋮	xshear = 0
		yshear = 0
		xforce = 0
		yforce = 0
		incremental = true

3.27.1 /distort/set/reset_params

Reset all the variables in this menu to their default value.

3.27.2 /distort/set/all_nodes

Apply the current distortion to all the nodes, whether or not they're in a nodegroup.

3.27.3 /distort/set/boundaries

Apply the current distortion to all nodes with a fixed degree of freedom. See /bc/fix (Section 3.22).

3.27.4 /distort/set/*nodegroup*

Apply the current distortion to the specific previously defined nodegroups. Note: only values for those degrees of freedom which were previously fixed by the /bc/fix Submenu will be affected by the current distortion.

3.27.5 /distort/set variables

xstrain Amount of strain in the x direction to apply. $x \to x \times (1 + \mathsf{xstrain})$. In other words, a uniform ϵ_{xx}. Default: 0.

ystrain Amount of strain in the y direction to apply. $y \to y \times (1 + \mathsf{ystrain})$. In other words, a uniform ϵ_{yy}. Default: 0.

xshift Amount to shift nodes in the x direction: $x \to x + \mathsf{xshift}$. Default: 0.

yshift Amount to shift nodes in the y direction: $y \to y + \mathsf{yshift}$. Default: 0.

xshear Amount of shear in the x direction to apply: $x \to x + \mathsf{xshear} \times y$. In other words, a uniform $\partial u_x/\partial y$, where u_x is the x–displacement. This is not the same as the xy component of the strain tensor, which is $(\partial u_x/\partial y + \partial u_y/\partial x)/2$. Default: 0.

yshear Amount of shear in the y direction to apply: $y \to y + \mathsf{yshear} \times x$. In other words, a uniform $\partial u_y/\partial x$. Default: 0.

xforce The x component of force to apply to the nodes. Default: 0.

yforce The y component of force to apply to the nodes. Default: 0.

incremental A boolean variable indicating whether the current distortions is absolute or incremental. Default: `true` (set incremental distortions).

3.28 /distort/clear

Each function in this menu removes the distortions from the nodes in a named nodegroup. The function all removes distortions from all nodes. By "remove" we mean that the incremental *and* absolute distortions are set to zero. To set only the absolute distortions to zero, use /reset in the main OOF menu. (Section 3.1.5.)

3.29 /distort/show

Show information about the current absolute and incremental distortions.

Menus	Functions	Variables
	commands	
	nodegroup...	
	⋮	

3.29.1 /distort/show/commands

Display (in the Message Window) all commands, chronologically, which have been invoked from the /distort/set Submenu.

3.29.2 /distort/show/*nodegroup*

Show (in the Message Window) the current state of the incremental and absolute distortions for the chosen nodegroup. The absolute distortions are the actual values at the nodes, and include the sum of all of the applied incremental distortions.

3.30 /output/tee

Most of the /output commands write results to the Message Window. Often you want to save results in a file. The commands in this menu let you open a file for the data and send some or all of the outpu to the file. (This menu is called "tee" because it acts like the Unix *tee* command, which is used to save a copy of data in a pipe.)

Menus	Functions	Variables
	open	
	close	
	show	

3.30.1 /output/tee/open

Start saving output results to a file. If another file is open, close it. Everything written to the message window falls into one of four categories: data, commands, info, and errors (see Section 1.5.2). The arguments let you select which kinds of output will appear in the file.

open arguments

filename The name of the file to open. Default: oof.out

data Whether or not to save data output. Default: true

commands Whether or not to save commands. Default: true

info Whether or not to save extra information. Default: false

errors Whether or not to save error messages. Default: false

append If true, the old contents of the file, if any, are preserved and new data is written at the end. If false, the new data overwrites the old data. Default: false

3.30.2 /output/tee/close

3.30.3 /output/tee/show

3.31 /output/grid

Get or save information about the current grid. This menu provides a way of saving the current state. The grid and the current values of all variables can be saved in a `.goof` file, which can be read in a separate OOF session.

Menus	Functions	Variables
matrix	ascii	
	binary	

The Submenus are:

matrix Prints information about the finite element stiffness matrices into files with fixed names. This is for debugging purposes only and can create very large ASCII files. Those familiar with finite element methods may find these files useful.

3.31.1 /output/grid/ascii

Print the grid and elements to a user-specified file in an ASCII format. This tends to create large files that are slow to read in later, so the binary option is preferred. However, the file can be viewed and information could be obtained from the rather simple format (described in Section 6.3.1).

ascii arguments

filename The name of the ASCII `.goof` file to be written. Default: ascii.goof

3.31.2 /output/grid/binary

Print the grid and elements to a user-specified file in an binary format. This tends to create smaller `.goof` files. The format of the binary file is described in Section 6.3.2.

binary arguments

filename The name of the binary `.goof` file to be written. This file can be subsequently compressed and read in compressed format, such as gzip. Default: binary.goof

save_stiffness Whether or not the local stiffness matrix for each element is saved in the data file. Computing the local stiffness matrices consumes a large fraction of the time spent while loading a data file, so saving the matrices significantly speeds up loading the file. However, it also significantly increases the file size. Default: false

3.32 /output/individual

Print information about each element individually. All functions in this menu and its submenus print data to a file. Each line of the file begins with an element index, followed by the quantity(s) of interest for that element. The element index can be used to select the element (see /select/elements/number, section 3.19.4), or to cross reference files created with different /output/individual functions. **Note that the element indices can change if the grid is modified and then equilibrated!**

Menus	Functions	Variables
stress	current_area	
strain	original_area	
	orientation	
	energy_density	

The Submenus are:

stress Print various stress components for each element. See section 3.33.

strain Print various strain components for each element. See section 3.34.

3.32.1 /output/individual/current_area

Save the area of each element in a file, taking into account the current distortions.

current_area arguments

filename The file in which to save the data.

3.32.2 /output/individual/original_area

Save the area of each element in a file, *without* taking into account the current distortions.

original_area arguments

filename The file to which to save the data.

3.32.3 /output/individual/orientation

Save the crystallographic orientation of each element in a file. See section 4.3 for an explanation of the format used for orientations.

orientation arguments

filename The file to which to save the data.

3.32.4 /output/individual/energy_density

Save the elastic energy density of each element in a file.

energy_density arguments

filename The file to which to save the data.

3.33 /output/individual/stress

Save the stress in each element in a file.

Menus	Functions	Variables
	tensor	
	eigenvals	

3.33.1 /output/individual/stress/tensor

Save the Cartesian components of the stress tensor in each element in a file. Each tensor is printed in the form `[xx xy xz; yy yz; zz]`.

tensor arguments

filename The file to which to save the data.

3.33.2 /output/individual/stress/eigenvals

Save the eigenvalues of the stress in each element in a file. All three eigenvalues are printed, in decreasing order.

eigenvals arguments

filename The file to which to save the data.

3.34 /output/individual/strain

Save the strain in each element in a file.

Menus	Functions	Variables
	all_comps	
	eigenvals	

3.34.1 /output/individual/strain/tensor

Save the Cartesian components of the strain in each element in a file. Each tensor is printed in the form `[xx xy xz; yy yz; zz]`.

tensor arguments

filename The file to which to save the data.

3.34.2 /output/individual/strain/eigenvals

Save the eigenvalues of the strain in each element in a file. All three eigenvalues are printed, in decreasing order.

eigenvals arguments

filename The file to which to save the data.

3.35 /output/force

Prints the total force on a nodegroup.

Menus	Functions	Variables
	nodegroup...	
	⋮	

3.35.1 /output/force/*nodegroup*

Prints each component of the total force acting on the specified nodegroup.

3.36 /output/displacement

Prints the average displacement on a nodegroup.

Menus	Functions	Variables
	nodegroup...	
	⋮	

3.36.1 /output/displacement/*nodegroup*

Prints the average value of each component of displacement for the specified nodegroup.

3.37 /output/stress

Prints the average stress for an element group.

Menus	Functions	Variables
statistics	selected	
	elementgroup...	
	⋮	

The Submenus are:

statistics Computes and reports stress statistics in several different formats. See section 3.38.

3.37.1 /output/stress/selected

Prints the average value of each component of the stress tensor of the currently selected elements in the Message Window. The stress is computed in the Cartesian coordinate system defined by the screen. The stress tensor is printed in the form `[xx xy xz; yy yz; zz]`.

3.37.2 /output/stress/*elementgroup*

Just like /output/stress/selected (3.37.1), but averages over an element group instead of the selected elements.

3.38 /output/stress/statistics

A menu designed to compute and print statistical quantities about the stress state of element groups. The type of information printed is controlled by the type variable.

Menus	Functions	Variables
	all	type = cartesian
	selected	
	elementgroup...	
	⋮	

3.38.1 /output/stress/statistics/all

Prints (in the Message Window) statistical information about the stress on all the elements.

3.38.2 /output/stress/statistics/selected

Prints (in the Message Window) statistical information about the stress on the currently selected elements.

3.38.3 /output/stress/statistics/*elementgroup*

Prints (in the Message Window) statistical information about the stress on the requested group of elements.

3.38.4 /output/stress/statistics

type A variable indicating the type of quantity to be computed. There are three choices: 1) `cartesian` calculates the mean and standard deviation of each component of the stress tensor as defined by the screen horizontal and vertical directions; 2) `eigenvalues` calculates the mean and standard deviation of the principal stress (the eigenvalues, labeled `emax`, `emid`, and `emin`) and the shear (computed as half the absolute value of the difference between the largest and smallest eigenvalues). These quantities are frame invariant; 3) `extremes` calculates the extreme (maximum and minimum) values of both Cartesian and invariant quantities. The means, deviations, and extremes of the stress components are printed in the tensor format used by /output/stress/selected (3.37.1), although the extremes don't really form a tensor. 4) `invariants` calculates the mean and standard deviation of the first, second, and third invariants of the stress.

3.39 /output/strain

Prints the average strain for an element group.

Menus	Functions	Variables
	selected	
	elementgroup...	
	⋮	

3.39.1 /output/strain/selected

Prints the average value of each component of the strain tensor of the currently selected elements in the Message Window. The strain is computed in the Cartesian coordinate system defined by the screen. The strain tensor is printed in the form `[xx xy xz; yy yz; zz]`.

3.39.2 /output/strain/*elementgroup*

Just like /output/strain/selected (3.39.1), but averages over an element group instead of the selected elements.

3.40 /output/cross_section

Plot a function along a given line in the xy plane. Because OOF can't (yet) create 2D graphs, these functions write data to a file, allowing you to use your own favorite plotting package. There are two submenus, one for uniformly spaced plot points, and one for non-uniformly spaced points. The non-uniformly spaced points are placed at element boundaries.

Both the uniform and nonuniform submenus contain a command for each of the **Graphics Drawers** (Sec. 5.1), although spaces are replaced with underscores in the command names. The arguments for the commands in the uniform submenu are listed in Sec. 3.40.1; the arguments for the nonuniform menu are listed in Sec. 3.40.2. In each submenu, all of the commands have the same arguments; they differ only in what function is plotted.

Both submenus look like this:

Menus	Functions	Variables
	Stress_XX	
	Stress_YY	
	Stress_ZZ	
	⋮	
	Strain_XX	
	Strain_YY	
	Strain_ZZ	
	⋮	
	Energy_Density	

3.40.1 /output/cross_section/uniform/*function*

Plot the given function along a given line in the xy plane, using uniformly spaced plot points.

cross_section/uniform arguments

filename Where the data will go.

start The (x, y) coordinates of the beginning of the line along which the data will be plotted. The coordinates must be given as a comma-separated pair in parentheses, *e.g*, `(1.2, 3.4)`. The coordinates are in the coordinate system of the screen, so to plot values relative to the coordinate system of the undistorted mesh, make sure to set /graphics/enhancement=0 (see sections 3.9.3 and 5.2.6).

finish The (x, y) coordinates of the end of the line.

npts The number of points at which to plot the function.

x Should the x coordinate of the plotted points be printed?

y Should the y coordinate of the plotted points be printed?

s Should the arclength coordinate of the plotted points be printed? The arclength is the distance from the starting point.

3.40.2 /output/cross_section/nonuniform/*function*

Plot the given function along a given line in the xy plane, using nonuniformly spaced plot points. The points will be placed within the specified tolerance of the element boundaries.

cross_section/nonuniform arguments

filename Where the data will go.

start The (x, y) coordinates of the beginning of the line along which the data will be plotted. The coordinates must be given as a comma-separated pair in parentheses, *e.g*, `(1.2, 3.4)`. The coordinates are in the coordinate system of the screen, so to plot values relative to the coordinate system of the undistorted mesh, make sure to set /graphics/enhancement=0 (see sections 3.9.3 and 5.2.6).

finish The (x, y) coordinates of the end of the line.

tolerance The minimum distance between plotted points.

x Should the x coordinate of the plotted points be printed?

y Should the y coordinate of the plotted points be printed?

s Should the arclength coordinate of the plotted points be printed? The arclength is the distance from the starting point.

Chapter 4

The Element Types

The local constitutive behavior of any part of the OOF model is incorporated in an element.

Element properties are specified in a function window, such as /modify/replace/hexagonal, illustrated in Figure 4.1. There is a separate function with its own set of parameters for each type of material. These functions each appear in two menus: /initialize/uniform creates a uniform grid of a given material, and /modify/replace replaces all selected elements with new elements of a given material.

The constitutive parameters derive from linear elasticity and thermoelasticity. Readers familiar with thermoelasticity can skip the background material in section 4.1.

Currently, each element must be treated in either *plane strain* or *plane stress*. The distinction between the two is discussed in section 4.2. Plane strain will be used if the variable planestrain in the function window is `true`.

Element orientations are discussed in Section 4.3.

4.0.1 Coordinate Conventions

The numerals 1, 2, and 3 are used to label the axes in a coordinate system attached to the crystalline axes. The letters x, y, and z refer to the lab (screen) coordinates: x is to the right, y is up, and z comes out of the screen. Unless a rotation is applied to a material via the orientation parameter (see Section 4.3), 1 aligns with x, 2 with y, and 3 with z.

4.1 Background Material: Basic Anisotropic Thermoelasticity

In the current version of OOF, all elements are triangles consisting of three *nodes* located at the triangle vertices. The nodes may be located at the boundary of the mesh, or they can be shared with other elements. The elements share information with each other (i.e., how much the corners have been displaced) through their nodes.

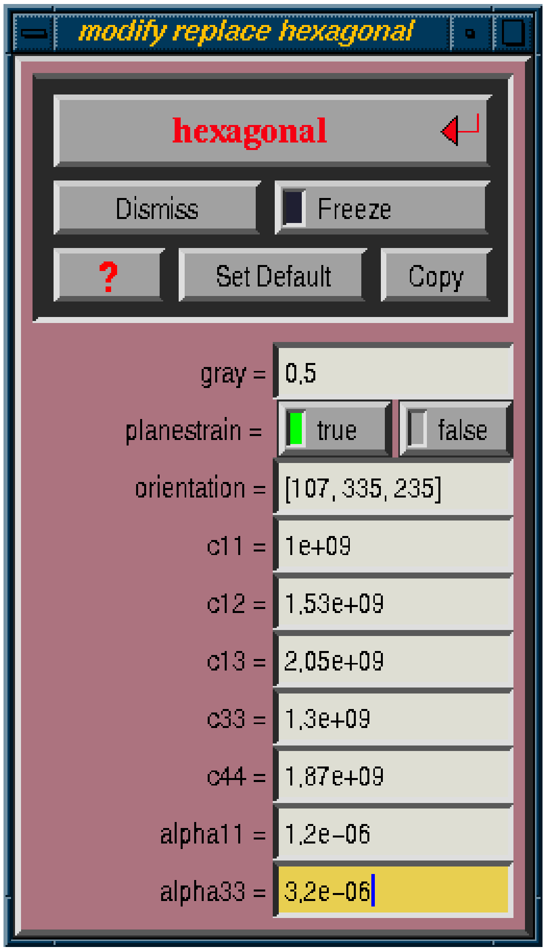

Figure 4.1: The function window for /modify/replace/hexagonal. All arguments other than gray are constitutive parameters. gray simply governs how the element will be displayed.

The current version of OOF is thermoelastic, and so the pertinent field quantities are the displacement (vector) field $\vec{u}$ and the uniform temperature change ΔT. The displacement field imparts a strain (tensor) field through the rotation-invariant values of spatial derivatives of the displacement field: $\epsilon_{ij} = (\partial u_i/\partial x_j + \partial u_j/\partial x_i)/2$–which is a long-winded way of saying the nodal displacements impose a strain, ϵ_{ij}, on the elements through a predictable formula.

The temperature is given as a difference ΔT. This difference is relative to the temperature at which the initial mesh is stress free. Each element undergoes a uniform strain, $\epsilon^T_{ij} = \alpha_{ij}\Delta T$, in the stress free state (i.e., if the nodes are completely unrestrained or the stress tensor $\sigma_{ij} = 0$). The second rank tensor α_{ij} is the (symmetric) thermal expansion tensor. It has additional symmetries according to the point group of the underlying material. See Nye [2] for a lucid discussion of how material symmetry is expressed in tensors representing linear constitutive behavior.

The stress σ_{ij} is a second rank tensor which represents the the total *force* transmitted in the $\hat{e}_i$-direction through a planar region with its normal parallel to the $\hat{e}_j$-direction in the limit that the area of the region shrinks to a point. Torque balance requires that $\sigma_{ij} = \sigma_{ji}$ in linear elasticity.

In linear elasticity, the stress is related to the strains implied by the displacement field and the stress-free strain[1] ϵ_{ij}^T through the linear relation:

$$\sigma_{ij} = C_{ijkl}(\epsilon_{kl} - \epsilon_{kl}^T)$$

The fourth-rank tensor C_{ijkl} is the *stiffness tensor*. The symmetry of both σ_{ij} and ϵ_{kl} imply that $C_{ijkl} = C_{ijlk} = C_{jikl}$, so it can have at most 36 independent components. An additional requirement, that the elastic energy density be frame invariant, imposes the symmetry $C_{ijkl} = C_{klij}$ and reduces the number of independent components to 21.

It is convenient to take advantage of the symmetries to represent the stiffness tensor as a symmetric 6×6 matrix. This can lead to *silly mistakes* since the matrix does not rotate in the usual straight-forward way. Fortunately, OOF handles the rotations for you (see Section 4.3). To convert the fourth-rank tensor into a two dimensional matrix, we adopt the convention in Nye [2] and replace the six distinct pairs ij with a single integer as follows:

Four Index Tensor Indices ij	11	22	33	23 or 32	31 or 13	12 or 21
Two Index Matrix Indices m	1	2	3	4	5	6

For example, the tensor components C_{1123}, C_{2311}, C_{1132}, and C_{3211}, which must all be equal by symmetry, are all represented by the matrix element C_{14}.

In OOF, the stiffness components are entered using the matrix notation. The symmetry of each particular material places additional constraints on the stiffness matrices. These additional symmetries are illustrated in Figures 4.2, 4.3, 4.4, and 4.5 (following Nye [2]).

4.2 Plane Stress and Plane Strain

Plane strain is defined by the strain state $\epsilon_{iz} = 0$; it is the limiting condition in the center plane of a very thick specimen. An out-of-plane stress σ_{iz} is required to maintain plane strain.

Plane stress is defined by the stress state $\sigma_{iz} = 0$; it holds at the surface of a thin specimen. The displacement field in the z direction is not zero nor is it uniform in an heterogeneous grid.

Using plane stress or plane strain, the stiffness tensor C_{ijkl} can be reduced from three to two dimensions, because the z components of stress and strain have been eliminated from the problem. In the matrix representation, C becomes a 3×3 matrix, with rows labeled 11, 22, and 12 in the original tensor indices.

[1] In the current version of OOF, only thermal stresses contribute to the stress-free strain, but this could be made more general.

4.3 Rotations

The orientation variable in figure 4.1 illustrates how the orientation of the material in an element is specified. In the general case, three angles are required to specify an orientation. We use the Euler angles to describe one rotation as the result of three simpler rotations. Historically, Euler angles have been the source of much confusion, so, as our contribution to history, we present multiple ways of thinking about them. *Different authors use different conventions, so don't assume that our definition is the same as yours!*

A rotation is entered as three values, in degrees, with square brackets: `[` α`,` β`,` γ `]`. If only one number is specified (without brackets), it is taken as the β value. That is, `30` is equivalent to `[0, 30, 0]` and is the same as rotating about an axis normal to the OOF screen. Note that `[0, 0, X]` is also equivalent to `[0, X, 0]`.

We define $(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}})$ to be the screen coordinate system, with $\hat{\mathbf{x}}$ pointing to the right, $\hat{\mathbf{y}}$ upwards, and $\hat{\mathbf{z}}$ out of the screen. We will call the crystal axes $(\hat{\mathbf{a}}, \hat{\mathbf{b}}, \hat{\mathbf{c}})$. As usual, all rotations are right handed, *ie.* a positive rotation about an axis is counterclockwise if the axis is pointing towards you.

One way of finding the Euler angles: This method describes the rotation that must be applied to the crystal axes to bring them into alignment with the screen axes, assuming that the crystal *starts* in its desired orientation with respect to the screen. First, rotate the axes by β around the $\hat{\mathbf{c}}$ axis. This defines new axes $(\hat{\mathbf{a}}', \hat{\mathbf{b}}', \hat{\mathbf{c}})$. Next rotate by α about the $\hat{\mathbf{b}}'$ axis, defining another coordinate system $(\hat{\mathbf{a}}'', \hat{\mathbf{b}}', \hat{\mathbf{c}}'')$. Finally rotate by γ about the $\hat{\mathbf{c}}''$ axis, bringing the axes into agreement with $(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}})$.

An equivalent way: Consider the material oriented with a globe, with the origin at the center of the earth and $\hat{\mathbf{c}}$-axis pointing towards the north pole, the $\hat{\mathbf{a}}$-axis pointing at the Greenwich Meridian where it intersects the equator and the $\hat{\mathbf{b}}$-axis pointing towards the Indian ocean somewhere southeast of Sri Lanka. The unrotated globe has its $\hat{\mathbf{a}}$, $\hat{\mathbf{b}}$, and $\hat{\mathbf{c}}$ axes aligned with the $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$ axes of the screen, respectively. The rotation triplet describes how to rotate the *material* into its desired orientation. The first number of the rotation triplet tilts the point on the north pole southward along the Greenwich Meridian by $-\alpha$ degrees to a new latitude. The second number spins the globe to the east by $-\beta$ degrees about its (tilted) $\hat{\mathbf{c}}$-axis. The third number rotates the tilted globe to the east by $-\gamma$ degrees about the screen's $\hat{\mathbf{z}}$ axis.

Yet another way: The order of rotations in the previous definition can be rearranged. Rotate the globe to the east by $-\beta$ degrees about the $\hat{\mathbf{z}}$-axis, tilt the north pole by $-\alpha$ about the $\hat{\mathbf{y}}$-axis, and rotate to the east by $-\gamma$ again about the $\hat{\mathbf{z}}$-axis.

A prescription: If you know the components in the crystal coordinate system of the screen vectors $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$, you can compute the Euler angles as follows. Let (x_1, x_2, x_3) be the components of the screen's $\hat{\mathbf{x}}$-axis in the crystal coordinate system. Let (y_1, y_2, y_3) and (z_1, z_2, z_3) similarly be the components of the $\hat{\mathbf{y}}$ and $\hat{\mathbf{z}}$-axes. Then

$$\begin{aligned}
\sin(\alpha) &= \sqrt{z_1^2 + z_2^2} & \cos(\alpha) &= z_3 \\
\sin(\beta) &= z_2/\sqrt{z_1^2 + z_2^2} & \cos(\beta) &= z_1/\sqrt{z_1^2 + z_2^2} \\
\sin(\gamma) &= y_3/\sqrt{z_1^2 + z_2^2} & \cos(\gamma) &= -x_3/\sqrt{z_1^2 + z_2^2}
\end{aligned}$$

4.4 Physical Units

The stiffness tensor has dimensions of stress (force divided by area). Most of the parameters used to specify material properties therefore are stresses. They can be entered in any units you wish, as long as you are consistent. The units you choose for stress, along with the units you choose for length, determine the units used for forces (as, for example, in Sections 3.27 and 3.35). The units of length are determined either by the height and width variables in the /initialize/uniform menu (Section 3.13) or by the width variable in PPM2OOF.

The thickness of the elements is assumed to be 1. Forces scale linearly with the thickness.

Thermal expansion coefficients have dimensions of strain/temperature, which is $(\text{temperature})^{-1}$ since strain is dimensionless. The units you use for thermal expansion must be consistent with those used for deltaT (Section 3.1).

In the listings in Section 4.5 below, the dimensions of each parameter are given in brackets at the end of each description. [like this]

4.5 Element Types and their Parameters

In this section we list each element type and its parameters. Two features are common to each element: the gray value and the boolean planestrain.

Parameters

gray The gray value only has an effect on how the element in a Mesh Drawer (see section 5.1) appears. It is a floating point number between 0 and 1, where 1 is white and 0 is black. [dimensionless]

planestrain If planestrain=`true`, a plane strain stiffness matrix is calculated for the element. The plane is that of the screen. If planestrain=`false`, then a plane stress stiffness is imposed on the element. [boolean]

4.5.1 isotropic

An isotropic element has the same stiffness independent of the loading direction or the plane on which the load is applied.

According to Figure 4.2, there are two independent elastic constants. OOF uses the Young's modulus E and the Poisson's ratio ν for the two input variables. In the case of plane stress, the 3×3 stiffness matrix is:

$$\frac{E}{1-\nu^2}\begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{pmatrix}.$$

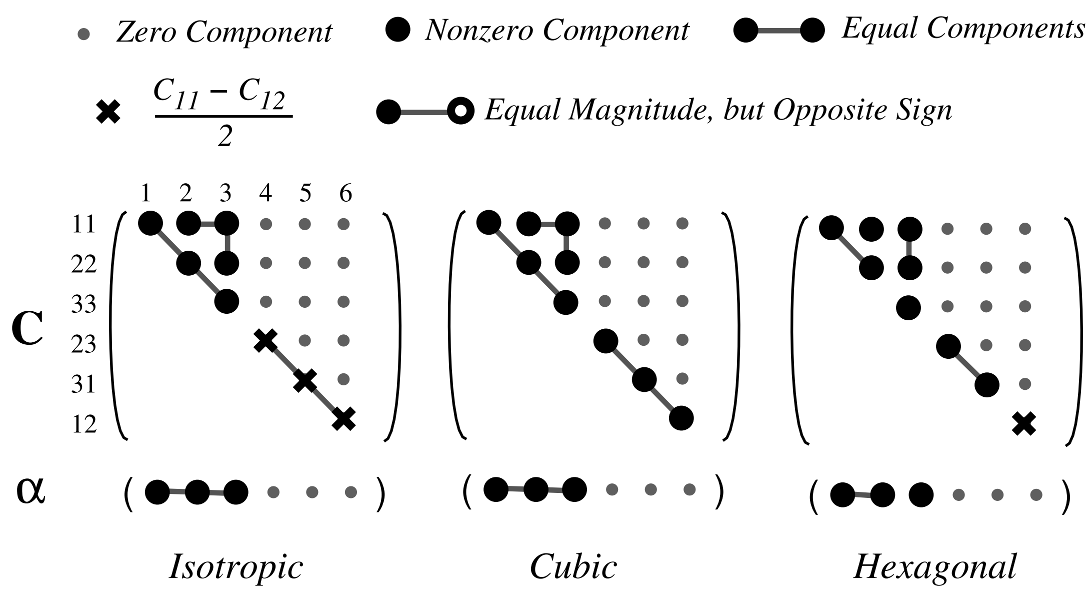

Figure 4.2: Symmetries of the stiffness matrix for Isotropic, Cubic and Hexagonal materials.

For plane strain, the stiffness matrix is:

$$\frac{E(1-\nu)}{(1+\nu)(1-2\nu)}\begin{pmatrix} 1 & \frac{\nu}{1-\nu} & 0 \\ \frac{\nu}{1-\nu} & 1 & 0 \\ 0 & 0 & \frac{1-2\nu}{2(1-\nu)} \end{pmatrix}.$$

Parameters

young The Young's modulus E [stress].

poisson The Poisson's ratio ν [dimensionless].

alpha The coefficient of thermal expansion [inverse temperature].

4.5.2 empty

Empty elements fill space, but do not have any stiffness. Empty elements can be used to make holes, or other regions of mesh which do not transmit any load.

4.5.3 cubic

These are elements with cubic symmetry. According to figure 4.2 there are three independent entries in the stiffness matrix. We pick those independent coefficients with a somewhat standard convention for the cubic Young's modulus (E), Poisson's ratio (ν) and the anisotropy factor ($A = 2C_{44}/(C_{11} - C_{12})$) which makes the 6×6 matrix form of C_{ijkl} look like:

$$\begin{pmatrix} \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} & \frac{E\nu}{(1+\nu)(1-2\nu)} & \frac{E\nu}{(1+\nu)(1-2\nu)} & 0 & 0 & 0 \\ \frac{E\nu}{(1+\nu)(1-2\nu)} & \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} & \frac{E\nu}{(1+\nu)(1-2\nu)} & 0 & 0 & 0 \\ \frac{E\nu}{(1+\nu)(1-2\nu)} & \frac{E\nu}{(1+\nu)(1-2\nu)} & \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{AE}{2(1+\nu)} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{AE}{2(1+\nu)} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{AE}{2(1+\nu)} \end{pmatrix}.$$

This is equivalent to the isotropic stiffness matrix when $A = 1$.

The two-dimensional stiffness matrix depends on whether `planestrain` is true or false.

Parameters

`young` The cubic Young's modulus E [stress].

`poisson` The cubic Poisson's ratio ν [dimensionless].

`anisotropy` The anisotropy factor, A [dimensionless].

`alpha` The coefficient of thermal expansion. In cubic materials $\alpha_{11} = \alpha_{22} = \alpha_{33}$; all off-diagonal terms vanish. [inverse temperature]

`orientation` The `orientation` of the crystalline axes. See Section 4.3. [degrees]

4.5.4 hexagonal

A material with hexagonal symmetry has 5 independent elastic constants and 2 independent coefficients of thermal expansion. See figure 4.2.

Parameters

`orientation` If unrotated (`orientation= 0`), a hexagonal element is isotropic in the xy plane (the plane of the screen), with the crystalline c-axis in the z direction (out of the screen). See Section 4.3. [degrees]

thermoelastic coefficients The components of the stiffness matrix are `c11`, `c12`, `c13`, `c33`, and `c44`. The two independent coefficients of thermal expansion are `alpha11` and `alpha33`.

4.5.5 orthorhombic

A material with orthorhombic symmetry has 9 independent elastic constants and 3 independent coefficients of thermal expansion. See figure 4.3.

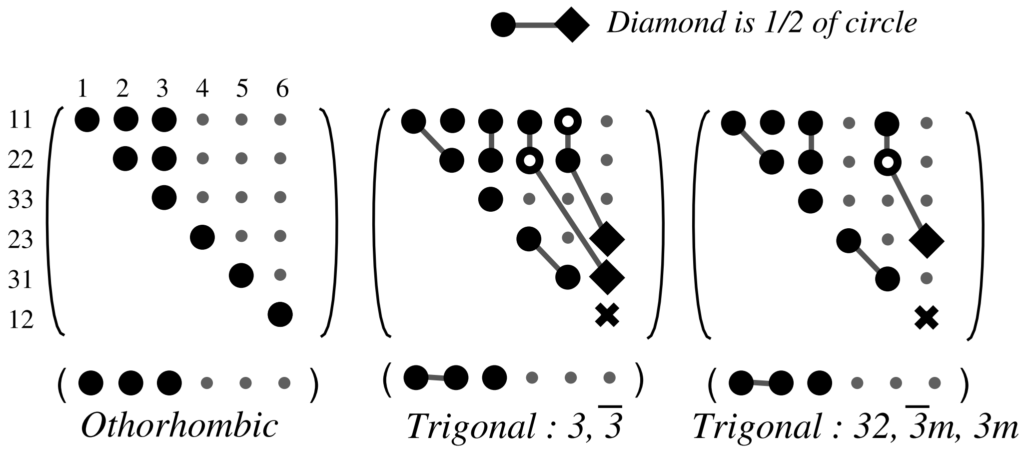

Figure 4.3: Symmetry relations for Orthorhombic classes and point groups in the Trigonal class. See Figure 4.2 for explanation of symbols.

Parameters

orientation The orientation of the crystalline axes. See Section 4.3. [degrees]

elastic coefficients The components of the stiffness matrix are specified by the values of c11, c12, c13, c22, c33, c44, c55, and c66. [stress]

thermal expansion coefficients The three independent coefficients of thermal expansion are specified by alpha11, alpha22, and alpha33. [inverse temperature]

4.5.6 eds_el

This element is named after Ed Fuller, who suggested it[2]. It is an element which is isotropic in its elastic coefficients, but is orthorhombic in its thermal expansion coefficients. It can be used to simulate transformation strains or residual stresses where the elastic coefficients are not known precisely, but the expansions are known not to be isotropic.

Parameters

elastic coefficients The elastic coefficients young and poisson have the same meaning as they do in the isotropic element (Section 4.5.1. [stress]

[2]There are only so many elements, so if you want one named after you, you should invent one and let us know soon.

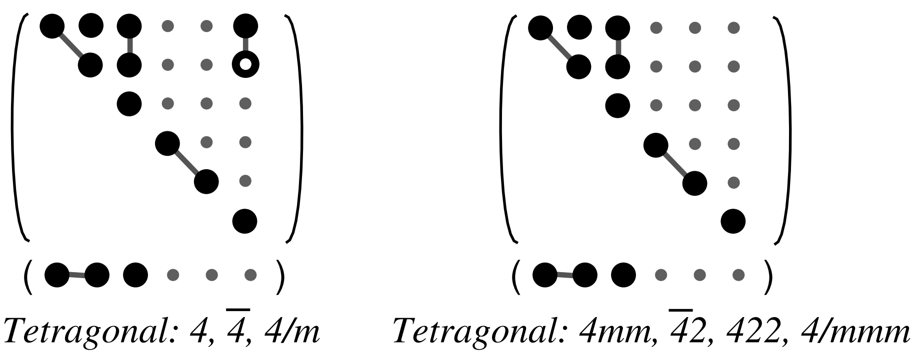

Figure 4.4: Symmetry relations for point groups in the Tetragonal class. See Figure 4.2 for explanation of symbols.

---

thermal expansion coefficients The thermal expansion coefficients a1, a2, and a3 are the diagonal components of the thermal expansion tensor α_{ij}. [3] [inverse temperature]

orientation Without rotation, the thermal expansion coefficient a1 governs expansion in the x direction, a2 in the y direction, and a3 in the z direction (out of the screen). See Section 4.3. [degrees]

4.5.7 trigonalA

A material with trigonal symmetry, with 7 independent elastic constants and two independent coefficients of thermal expansion. It represents the crystal symmetries (3) and ($\bar{3}$). See figure 4.3, and also Sections 4.5.8 and 4.5.12.

Parameters

orientation The orientation of the crystalline axes. See Section 4.3. [degrees]

elastic coefficients The components of the stiffness matrix are specified by the values of c11, c12, c13, c14, c15, c33, and c44. [stress]

thermal expansion coefficients The three independent coefficients of thermal expansion are specified by alpha11 and alpha33. [inverse temperature]

[3] a1, a2, and a3 are the same as alpha11, alpha22, and alpha33 in the orthorhombic element, but are named differently for no good reason.

4.5.8 trigonalB

A material with trigonal symmetry, with 6 independent elastic constants and two independent coefficients of thermal expansion. It represents the crystal symmetries (3 2), (3 m), and ($\bar{3}m$).[4] It is the same as trigonalA with c15 = 0. See figure 4.3, and also Sections 4.5.7 and 4.5.12.

Parameters

orientation The orientation of the crystalline axes. See Section 4.3. [degrees]

elastic coefficients The components of the stiffness matrix are specified by the values of c11, c12, c13, c14, c33, and c44. [stress]

thermal expansion coefficients The three independent coefficients of thermal expansion are specified by alpha11 and alpha33. [inverse temperature]

4.5.9 damisotropic

This is an isotropic element which can undergo "damage" when the principal stress exceeds a user-defined upper limit or retrocedes below a user-defined lower limit. The effect of damage is that all components of the stiffness matrix are multiplied by a user-defined *knockdown* value.

The damage does not take place until a mutation command is invoked, either explicitly with mutate (Section 3.1.4) or implicitly with the comp_equil (Section 3.1.3) command. An element can be re-mutated (*i.e*, re-multiplied by the knockdown factor) if the stress re-exceeds its bounds.

Elements which have mutated are drawn in black in the **Mesh Drawer** (Section 5.1), but are immediately redrawn with the new (unequilibrated) stresses or strains in the **Stress Drawers** and **Strain Drawers**. An additional equilibrate step is required to produce equilibrium after any elements incur damage due to a mutate command.

Parameters

thermoelastic coefficients The thermoelastic coefficients young, poisson, and alpha are identical to those defined for the isotropic elements in Section 4.5.1.

max_stress The upper limit of stress. Elements whose largest principal stress is greater than max_stress will mutate during a mutate command (Section 3.1.4). [stress]

min_stress The lower limit of stress. Elements whose smallest principal stress is less than min_stress will mutate during a mutate command (Section 3.1.4). Negative stresses are compressive. [stress]

[4] The reference works are somewhat confusing. ($\bar{3}m$) may be the same as ($\bar{3}$ $2/m$). If you know for sure, please tell us.

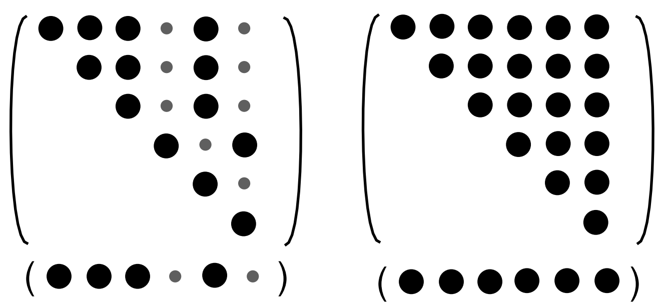

Figure 4.5: Symmetry relations for the Monoclinic and Triclinic classes. See Figure 4.2 for explanation of symbols.

---

max_knockdown The factor by which the entire element stiffness matrix is multiplied when its largest principal stress exceeds max_stress. [dimensionless]

min_knockdown The factor by which the entire element stiffness matrix is multiplied when its smallest principal stress is less than min_stress. [dimensionless]

4.5.10 damage

This is an isotropic element with more control over the way damage is incurred. See Section 4.5.9 for discussion on mutating elements and damage.

The idea is that the stiffness drop in the direction of maximum tension may be different than the stiffness drop associated with other directions. For instance, if a brick develops a crack it can still support loads in directions which are not normal to the crack plane.

There is one user-defined upper bound and two user-defined knockdown factors.

The process of mutation is the following:

1. The maximum principal stress (eigenvalue) is calculated and compared to the specified maximum stress.

2. If the maximum specified stress is exceeded, the direction of the principal stress axis is determined. (This direction may be out of the plane.) The stiffness matrix is rotated into a new coordinate system in which the principal stress lies along the z axis.

3. In the rotated coordinate system, C_{zzzz} is multiplied by knockdown1. All other components of C_{ijkl} with at least one of ijkl=z are multiplied by knockdown2.

4. The stiffness matrix is rotated back into its original orientation.

Parameters

orientation The orientation must be specified, but it has no effect since the thermoelastic coefficients are isotropic before mutation. The orientation is calculated in the mutation step, but the element is rotated and then *rotated back*, so the orientation after mutation contains no useful information either.[5] The orientation of the principal stress axis is reported in the message window as elements mutate. [degrees]

thermoelastic coefficients All thermoelastic coefficients for a undamaged damage element are isotropic as described in section 4.5.1.

maxstress The value to which the maximum principal stress is compared to see if the element should incur damage when the mutate command is invoked. See comments in section 4.5.9. [stress]

knockdown1 The factor which multiplies the rotated stiffness matrix's C_{zzzz} component during a mutation. [dimensionless]

knockdown2 The factor which multiplies the rotated stiffness matrix's remaining C_{ijkl} components for at least one of ijkl=z during a mutation. [dimensionless]

only_once If `true`, an element which has been damaged during one mutate command will not be damaged further on subsequent mutate commands. If `false`, the knockdown factors may be applied multiple times. Default: `false`

4.5.11 griffith and griffith2

These elements are designed to fail (*i.e,* mutate (Section 3.1.4)) under the Griffith criterion, which states that a crack will propagate when the total surface energy required to propagate the crack can be supplied by the elastic energy stored in the body:

$$2\ell\gamma < \frac{1}{2}\int \sigma\epsilon\, dV, \tag{4.1}$$

where ℓ is the crack length and γ is the surface energy of the cracked interface. In OOF the element size is used to specify the characteristic crack length and the volume of integration is the element area (per unit depth). In other words, the elastic energy is assumed to come entirely from the cracking element.

Because elements in OOF can't actually crack, after mutating the elements reduce their moduli anisotropically to simulate a crack, and the amount of surface area generated by mutation has to be estimated. We take it to be the diameter of a circle with the same area as the element.

CAUTION: Neither griffith nor griffith2 is a fully correct implementation of the Griffith criterion. griffith mutates when the stored elastic energy before mutation is greater than

[5] *I.e.,* the orientation of the anisotropic mutated element is encoded in the stiffness matrix, not stored in the orientation parameter.

the surface energy after mutation, which doesn't account for the fact that there is still some elastic energy left in the element after it mutates. griffith2 solves this problem partially by not including energy due to stress perpendicular to the surface.

The process of mutation is similar to that for the damage element (Section 4.5.10):

1. The energy balance for cracking is computed.
2. If the energy balance is favorable for cracking, the direction of the maximum principle stress is found.
3. Modfication of the stiffness matrix proceeds as in the damage element (Section 4.5.10).

Parameters

orientation As with the damage element, this is meaningless. See the explanation in Section 4.5.10. [degrees]

thermoelastic coefficients All thermoelastic coefficients for a undamaged damage element are isotropic as described in section 4.5.1.

gamma The energy per unit length of the crack surface. [stress/length][6]

knockdown1 The factor which multiplies the rotated stiffness matrix's C_{zzzz} component during a mutation. This should be a small number, if the element is really to be "cracked". [dimensionless]

knockdown2 The factor which multiplies the rotated stiffness matrix's remaining C_{ijkl} components for at least one of ijkl=z during a mutation. [dimensionless]

only_once If `true`, an element which has been damaged during one mutate command will not be damaged further on subsequent mutate commands. If `false`, the knockdown factors may be applied multiple times. Default: `false`

4.5.12 trigonalCrack

A trigonalA element (Section 4.5.7) with the griffith2 fracture criterion (Section 4.5.11).

Parameters

orientation The orientation of the trigonal element before it mutates. See Section 4.3.

thermoelastic coefficients Before mutation, the components of the stiffness matrix are those of the trigonalA element (Section 4.5.7): c11, c12, c13, c14, c15, c33, and c44. The two independent coefficients of thermal expansion are alpha11 and alpha33.

[6] Energy and stress have the same units, and the thickness of the sample is taken to be 1.

gamma The surface energy of the crack.

knockdown1 The factor which multiplies the rotated stiffness matrix's C_{zzzz} component during a mutation. [dimensionless]

knockdown2 The factor which multiplies the rotated stiffness matrix's remaining C_{ijkl} components for at least one of ijkl=z during a mutation. [dimensionless]

only_once If `true`, an element which has been damaged during one mutate command will not be damaged further on subsequent mutate commands. If `false`, the knockdown factors may be applied multiple times. Default: `false`

4.5.13 zimmer and zimmer2

André Zimmermann asked for this element, so we put his name on it. It's a griffith element, but before it mutates it has hexagonal symmetry. It has the same problems that the griffith element does. See Section 4.5.11. Zimmer2 differs from zimmer in the same way that griffith2 differs from griffith: the zimmer element mutates if the total stored elastic energy exceeds the surface energy cost of a crack whereas the zimmer2 element mutates if the elastic energy due to the maximum principal stress exceeds the surface energy.

Parameters

orientation The orientation of the hexagonal element before it mutates. If unrotated (orientation= 0), a hexagonal element is isotropic in the xy plane (the plane of the screen), with the crystalline c-axis in the z direction (out of the screen). See Section 4.3.

thermoelastic coefficients Before mutation, the components of the stiffness matrix are those of the hexagonal element (Section 4.5.4): c11, c12, c13, c33, and c44. The two independent coefficients of thermal expansion are alpha11 and alpha33.

gamma The surface energy of the crack.

knockdown1 The factor which multiplies the rotated stiffness matrix's C_{zzzz} component during a mutation. [dimensionless]

knockdown2 The factor which multiplies the rotated stiffness matrix's remaining C_{ijkl} components for at least one of ijkl=z during a mutation. [dimensionless]

only_once If `true`, an element which has been damaged during one mutate command will not be damaged further on subsequent mutate commands. If `false`, the knockdown factors may be applied multiple times. Default: `false`

Chapter 5

Graphics Drawers

This section describes the *Graphics Drawers*. Drawers are opened by /graphics/open. For an overview of their features, see Section 1.5.4 in the Introduction. In summary, what is being drawn is controlled by the **Drawing Selector**. The behavior of the mouse in the **Drawing Area** depends on which **Dashboard** is open. The available **Dashboards** depend on what's being drawn. Use your window manager to change the size and shape of the Drawer if you want a different size **Drawing Area**.

This chapter describes many ways of changing what's being drawn. If the image is large and takes a while to draw, and if you are changing many attributes of the image, you can click on the **Hold** button. This button inhibits redrawing until you click on it again, so that you don't have to wait for the image to be redrawn each time you change an attribute. The **Drawing Selector** cannot be held. While the image is held, it cannot be scrolled.

For obscure technical reasons a few aspects of the graphical output are controlled from the main graphics menu (Sec. 3.9) instead of from the drawers. These include the wallpaper variable, controlling the graphics background, and the enhancement variable, controlling how the mesh distortion is displayed.

5.1 The Drawers

When a Drawer is first opened, the **Mesh Drawer** will be displayed. The Mesh Drawer displays the finite element mesh, with the elements colored according to their gray value (see Chapter 4). Elements and nodes can be selected in the Mesh Drawer.

The rest of the drawers color the elements according to the local stress and strain. See Section 5.2.3 to interpret and control the color scheme.

The Drawers **Stress XX**, **Stress YY**, **Stress ZZ**, **Stress XY**, **Stress XZ**, and **Stress YZ** display the Cartesian components of the Stress (*i.e,* the stress in the screen coordinate system.)

The Drawer **Stress invariant 1** displays the first invariant (the trace) of the stress, $\sigma_{xx} + \sigma_{yy} + \sigma_{zz}$, which is the pressure. The Drawer **Stress invariant 2** displays the second invariant,

$$\sigma_{xx}\sigma_{zz} + \sigma_{xx}\sigma_{yy} + \sigma_{yy}\sigma_{zz} - \sigma_{xy}\sigma_{xy} - \sigma_{xz}\sigma_{xz} - \sigma_{yz}\sigma_{yz} \tag{5.1}$$

The Drawer **Stress invariant 3** displays the third invariant, or determinant, of the stress tensor.

The Drawers **Stress principal 1**, **Stress principal 2**, and **Stress principal 3** display the largest (1), middle (2) and smallest (3) eigenvalues of the stress.

The Drawer **Shear Stress** draws the invariant shear stress, which is one half the difference between the maximum and minimum eigenvalues of the stress.

Each of the above-mentioned **Stress** Drawers has a corresponding **Strain** Drawer.

The Drawer **Energy Density** draws the elastic energy density.

5.2 The Dashboards

The Dashboards display information about the image and determine what happens when the mouse is clicked in the **Drawing Area**.

5.2.1 Coordinates Dashboard

The **Coordinates** Dashboard (Figure 5.1) determines what part of the grid is visible in the **Drawing Area**.

The physical coordinates of the corners of the visible image are printed at the bottom of the Dashboard.

Clicking the right mouse button in the image zooms in around the point of the click. That is, the clicked point in the image stays (more or less) under the mouse, but the size of the image increases by the **Zoom Factor**. The **Zoom Factor** can be set in the input field in the dash board. Similarly, clicking on the image with the left mouse button zooms out—the image shrinks in the window.

The image can be moved within the **Drawing Area** by clicking and dragging with the middle mouse button.

The button labeled **Show Entire Image** shifts and scales the image so that it is all visible. The button labeled **Redraw** redraws the image, if it has somehow become corrupted.

When the image has been zoomed or shifted so that not all of it is visible, the **Scroll Bars** along its top and left edges become active. Moving the scroll bars with any mouse button shifts the image. The scroll bars are available no matter which Dashboard is open.

Clicking on the **Zoom Button** at the upper left corner of the **Drawing Area** is subtly different than clicking in the **Drawing Area** itself. Right and left mouse clicks on the **Zoom** button zoom in and out while keeping the point at the center of the window fixed. The middle mouse button clicked on the **Zoom** button duplicates the behavior of the **Show Entire Image** button. The **Zoom** button is also available no matter which Dashboard is open.

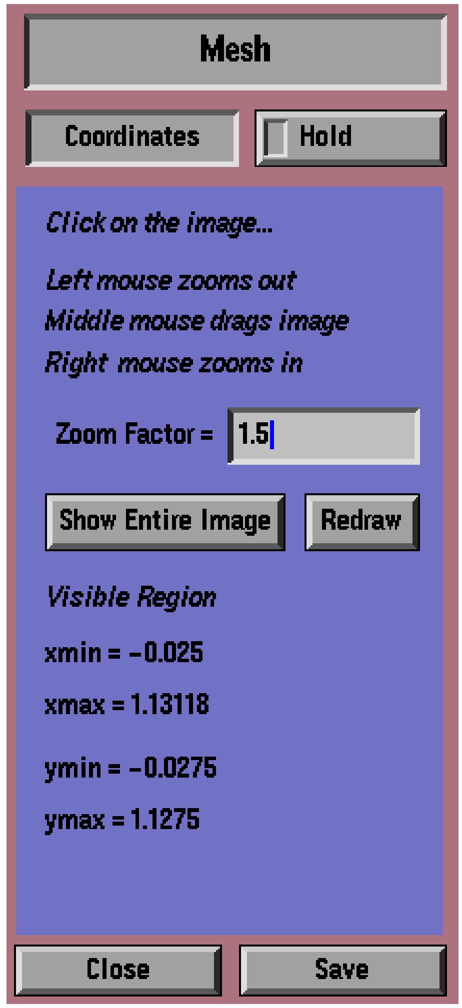

Figure 5.1: The **Coordinates** Dashboard.

5.2.2 Attributes Dashboard

The **Attributes** Dashboard (Figure 5.2) determines which features of the image are visible. It has two forms, depending on whether it's being used in a **Mesh** Drawer or not. In all cases, it has a bunch of check boxes which can be turned on and off by clicking on them:

Edges Whether or not to draw the edges of the finite elements. The width (in pixels) of the lines to use can be entered in the **width** box. A width of 0 does not mean to draw no line, rather it tells the X server to use the fastest algorithm for drawing a line of width 1, possibly making a few mistakes.

Elements Whether or not to draw the interiors of the elements.

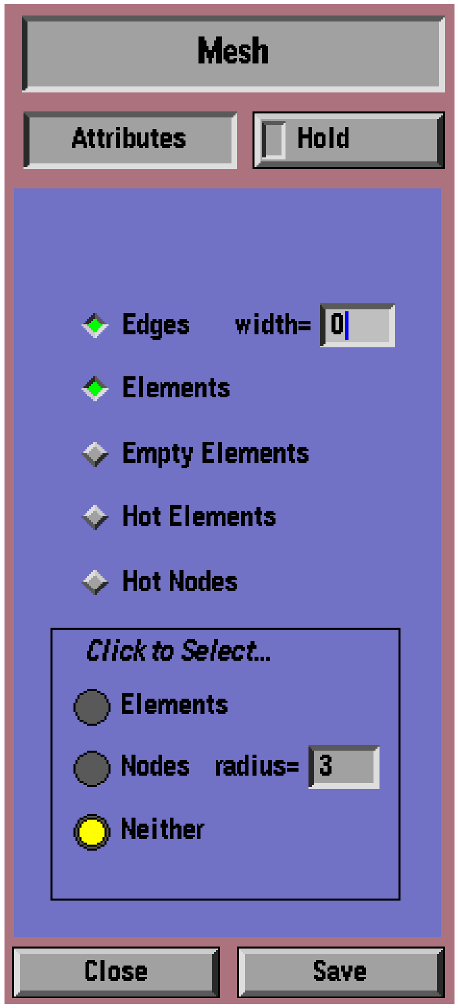

Figure 5.2: The **Attributes** Dashboard as it appears in the Mesh Drawer. Edges and Elements are being drawn, while Empty Elements and selections are not.

Empty Elements Whether or not to draw the Empty elements. Empty elements have a gray value, so they can be drawn in the **Mesh** Drawer. They always have zero stress when drawn in the **Stress** Drawers, but they may have non-zero strain. Be aware that nodes that are only contained in empty elements do not move when the grid is equilibrated, so the strain may appear to be discontinuous and the mesh overlapping when empty elements are drawn.

The following check boxes only appear in the **Mesh** Drawer:

Hot Elements Whether or not to highlight the selected elements.

Hot Nodes Whether or not to highlight the selected nodes. Highlighted nodes. Nodes are highlighted by drawing a small circle on top of them; if there are a lot of nodes, the circles can run into one another and the picture can be very ugly, as well as taking a long time to draw.

In the **Mesh** Drawer, elements and nodes can be selected with the mouse when the **Attributes** Dashboard is open. One of the three buttons labeled **Click to Select...** at the bottom of the Dashboard is always highlighted.

When the **Neither** button is highlighted, mouse clicks in the **Drawing Area** are ignored.

When **Elements** is highlighted, elements can be selected. (Notice that turning on this buttons automatically turns on the **Hot Elements** check box.) Clicking on a single element in the **Drawing Area** selects and highlights it while unselecting all others. Holding down the shift key while clicking toggles the selection of the clicked element while leaving all other elements alone. Clicking and *dragging* the mouse traces out a rectangle, and selects all elements with centers within the rectangle. Holding the shift key while dragging the mouse toggles the selection of all elements within the rectangle.

When **Nodes** is highlighted under **Click to Select...**, mouse clicks in the **Drawing Area** select nodes in a completely analogous way. The radius of the circle used to mark the selected nodes can be set in the box by the **Nodes** button. *Note that when shift-clicking to unselect nodes, the circle does not go away. This is a bug. Use the* **Redraw** *button in the* **Coordinates** *Dashboard to refresh the image and draw the correct circles.*

5.2.3 Color Dashboard

The **Color** Dashboard (Figure 5.3) controls the color scheme used to display stresses and strains in the various **Stress** and **Strain** Drawers. Because the form of the **Color** Dashboard is identical, no matter what's being displayed, in this section we'll call the displayed quantity *stress* to avoid excess, extraneous, and superfluous verbiage.

The bar on the left side is the **color bar**; it displays all of the colors used in the image.

The maximum and minimum values of stress in the grid are printed in the region labeled **image limits** in Figure 5.3. These values correspond to the range of colors displayed in the Drawer if the **Full Scale** check box is highlighted. If **Full Scale** is not highlighted, then the **limit** boxes at the ends of the **color bar** indicate the values of stress corresponding to the colors at the ends of the bar. These limits can be changed by typing new values into the **limit** boxes. (Make sure to hit ⟨*return*⟩ after typing a new value, or it won't take effect.) The **limit** boxes do not appear if **Full Scale** is on.

The *color scheme* determines how values of stress are converted into colors. The name of the current color scheme is shown in the **Scheme Selector**. Use the left mouse button on the **Scheme Selector** to bring up a menu of available color schemes, or use the middle and right buttons to cycle through the schemes. The number of colors used in the scheme can be set by typing in the **Size** box. A large number of colors (say 100) will give smooth gradations, but smaller numbers (10 or 20) can be useful for visualizing stress contours. On

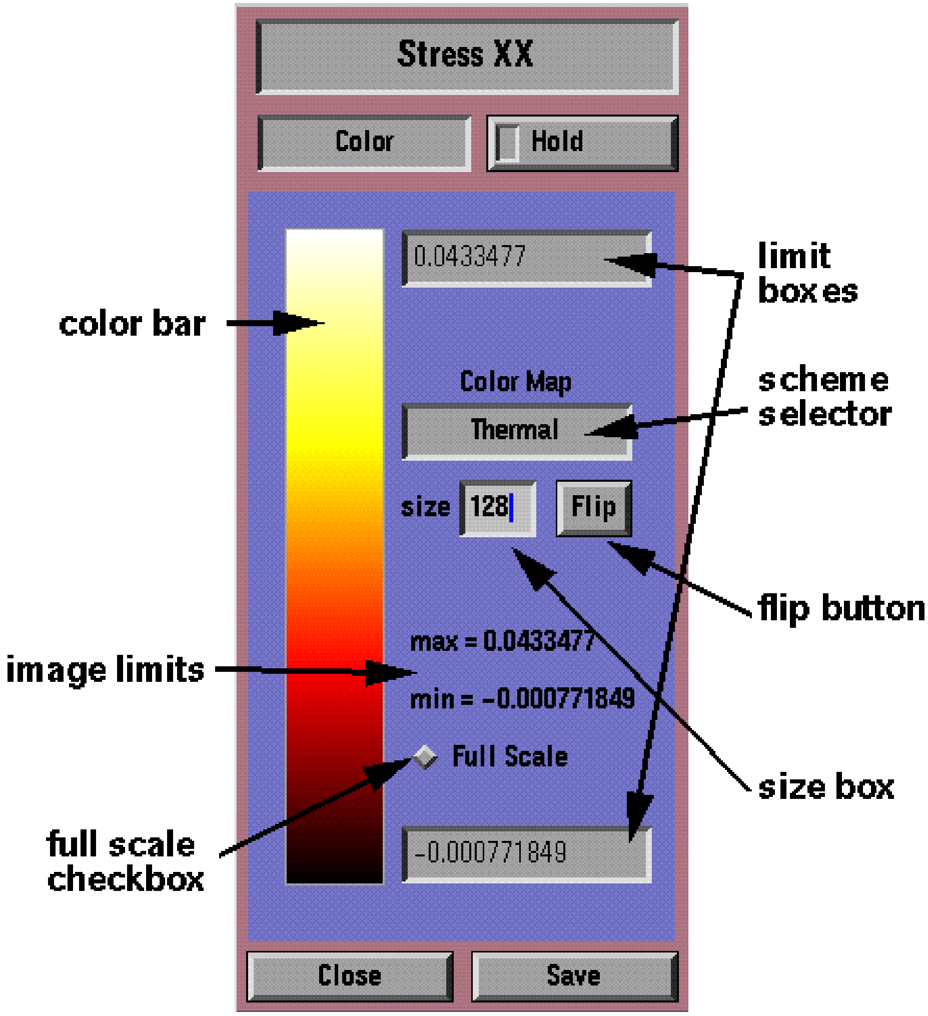

Figure 5.3: Anatomy of the **Color** Dashboard.

machines with 8-bit graphics, attempting to use too many colors is a mistake. The **Flip** button inverts the color scheme.

The **Color** Dashboard includes a primitive way of editing the colors in the current color scheme. This can be handy if you'd like to highlight a particular value of stress in the image. Clicking on a color in the color bar with the *right* mouse button will print the color and the stress range that it represents in the Message Window. Clicking with the *left* mouse button brings up a **Color Browser** (Figure 5.4) which allows the color to be changed. Clicking on the color bar with the *middle* mouse button makes the chosen color revert to its default value in the current color scheme. *Note that the changes in the color scheme are lost if you change Drawers!*

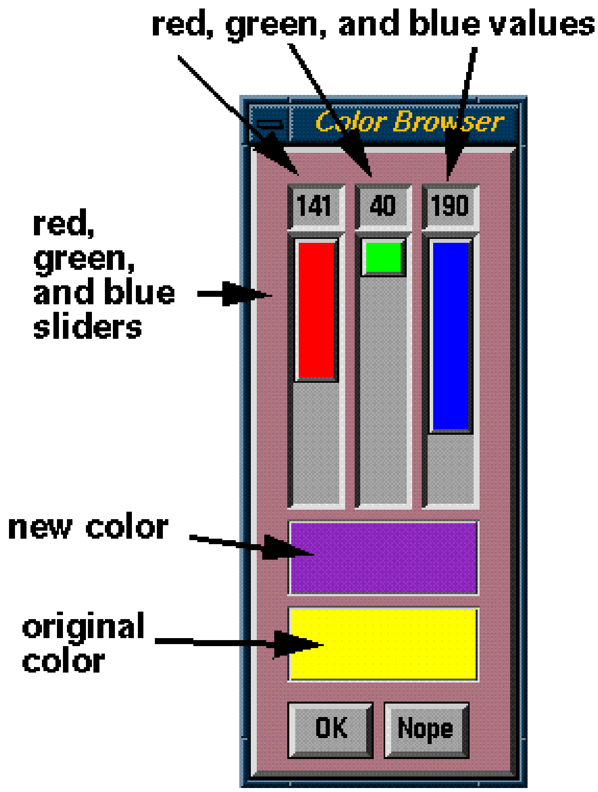

Figure 5.4: Anatomy of the **Color Browser**. The **original color** is the color that's being changed. Clicking on the **OK** button changes the original color to the **new color**, and clicking on the **Nope** button closes the window without changing the color. The **new color** is set by clicking and dragging the mouse on the red, green and blue **sliders**. The numbers above the sliders display their current values, in the range 0 to 255. If the sliders don't give you fine enough control, it's because they are less than 255 pixels long. Resizing the window should help.

5.2.4 Element Info Dashboard

When the **Element Info** Dashboard is open, clicking the mouse on an element in the image prints information about the element to the Message Window. When the mouse is over the **Drawing Area** the cursor changes to a question mark. **The element being queried is under the dot at the bottom of the question mark.**

The check boxes in the Dashboard determine what information is printed.

mouse position The coordinates of the mouse in physical units.

element type The type of element, as described in Chapter 4.

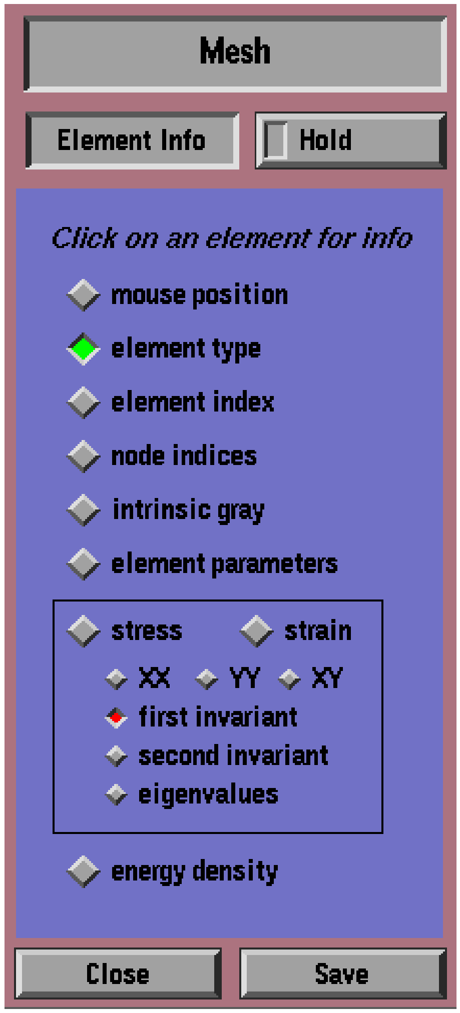

Figure 5.5: The **Element Info** Dashboard.

---

element index The index of the element. Each element has a unique integer index, which can be used to identify it in the output files created by the /output/individual commands (Section 3.32). Note that the element index can change if the grid is modified and then equilibrated!

node indices The indices of the nodes at the corners of the elements. These indices may be used in the command /output/node, for example. They are also used internally in the `.goof` file.

intrinsic gray The gray value of the element, as described in Chapter 4.

element parameters Again, see Chapter 4. These are the arguments to the function that was used to create the element.

stress and **strain** The stress and/or strain in the element. Which components of stress and strain are printed depend on the settings of the smaller check boxes below. Note that strain can be queried even in a **Stress** Drawer, and vice versa.

energy density The elastic energy density in the element.

The check boxes determining which components of stress and strain are printed are:

components The Cartesian components in the screen's coordinate system.

invariants The first, second, and third invariants, as described in Section 5.1.

eigenvalues The eigenvalues, as well as their orientations. Orientations are either a single number or a pair in square brackets. The pair (θ, ϕ) is the standard representation of an angle on the unit sphere; *i.e*, θ is the declination from the z axis in degrees, and ϕ is the rotation about the z axis, as measured counterclockwise from the x axis. If $\phi = \pi/2$, then only θ is given.

5.2.5 Node Info Dashboard

When the **Node Info** Dashboard is open, clicking the mouse on a node in the image prints information about the node to the Message Window. When the mouse is over the **Drawing Area** the cursor changes to a question mark. The node being queried is the closest one to the hook of the question mark.

The check boxes in the Dashboard determine what information is printed.

mouse position The coordinates of the mouse in physical units.

node type The type of the node, as described in Section 3.13.1 under the discussion of the interior_nodes variable. In addition, the status of the node's x and y degrees of freedom is reported. `fixed(`n`)` means that the degree of freedom is fixed (Section 3.22) and will not move during equilibration. The number n in parenthesis is the number of fixed nodegroups to which this degree of freedom belongs. `free` means that the degree of freedom is not fixed. `slaved` means that the degree of freedom is slaved to others in its nodegroup (Section 3.24). `autonomous` means that the degree of freedom is not enslaved. It is possible for a degree of freedom to be simultaneously free and enslaved. Sorry about that.

node index The index of the node. This may be used in the command /output/node. Node indices are also used internally in the `.goof` file.

current position The position of the node, called `x, y` in the Message Window.

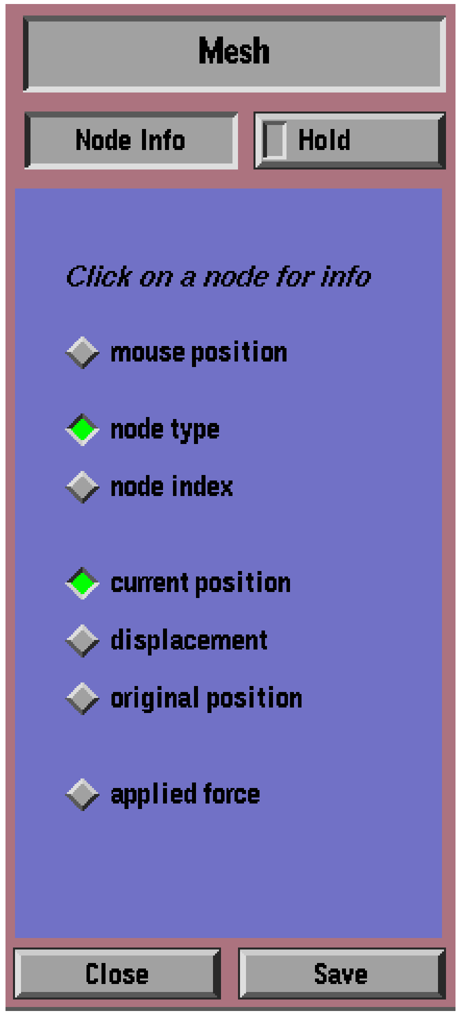

Figure 5.6: The **Node Info** Dashboard.

---

displacement The displacement of the node from its original position, called `dx, dy` in the Message Window.

original position The position of the node before the grid was distorted or equilibrated, called `x0, y0` in the Message Window.

applied force The force applied to the node via the /distort/set menu (Section 3.27), called `fx, fy` in the Message Window. Note that if the node contains an enslaved degree of freedom, then the reported force is the force on the master degree of freedom, which is the sum of the forces on all of the master's slaves. The force reported for a fixed degree of freedom is NOT the force reported by /output/force (Section 3.35), although it should be.

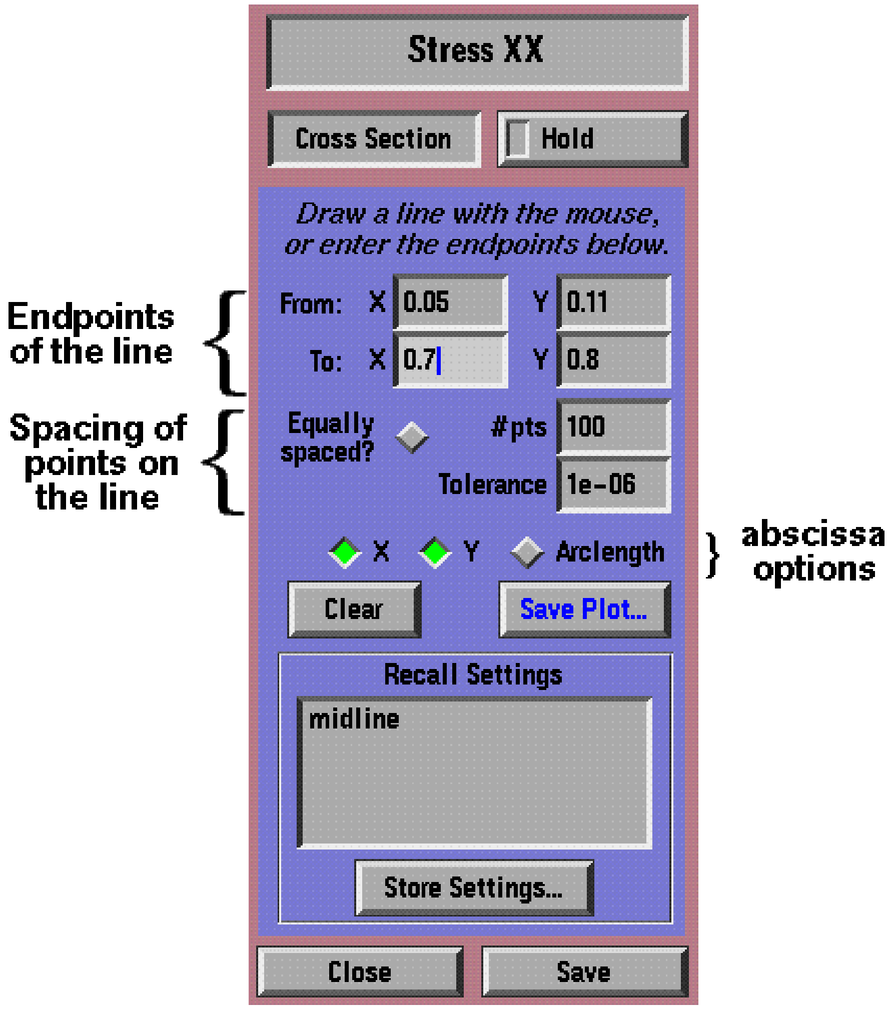

Figure 5.7: Anatomy of the **Cross Section** Dashboard.

5.2.6 Cross Section Dashboard

The **Cross Section** Dashboard is for plotting the value of the displayed function along an arbitrary line in the x-y plane. The plots don't appear on the screen. They are saved in a file, so that you can use your favorite plotting program. The cross section dashboard appears in every Drawer for which it makes sense to plot something. It doesn't appear in the Mesh Drawer.

Clicking and dragging the mouse on the image draws a line on the screen. The function will be plotted along the straight line going from the point where the mouse was clicked to the point where it was released. The endpoints of the line are displayed in the dashboard; they may be edited there.

How the plotted points are distributed along the line is determined by the **Equally spaced?** button, the **#pts** value, and the **Tolerance**. If **Equally spaced?** is lit, then the specified number of points (**#pts**) are evenly distributed along the line. If the button is not lit, then points are placed only at the element boundaries, the minimum distance between points will be less than or equal to **Tolerance**, and **#pts** is not used. (The functions being plotted are piecewise constant because of the kinds of elements that OOF uses, so the most efficient plot places points only at the discontinuities.)

The three buttons marked **X**, **Y**, and **Arclength** control what is plotted in addition to the function values. Arclength is the distance from the starting point of the plot. Positions are measured in the coordinate system of the image, so if you want them to be in the coordinate system of the undistorted system, make sure to set enhancement (Sec. 3.9.3) to zero.

The **Clear** button removes the plot line from the image and clears the input fields for the endpoints.

The **Save Plot...** button brings up a file selector and saves the data points in the specified file.

The current settings (endpoints, number of points, *etc.*) can be stored by clicking on the **Store Settings...** button. A window will appear asking you to give a name to the set of settings, and that name will appear in the list entitled **Recall Settings**. Clicking on a name in that list will restore the settings.

Chapter 6

Technical Notes

6.1 The Text Interface

All of the commands in the OOF **Menus** can be executed from a text interface. Executing text commands is very handy if you want to automate a process, or run OOF when logged in from home over a slow telephone line. The format for the text commands is the same, whether commands are being typed interactively, are being read from a log file or command file, or being executed as macros or as arguments to the `loop` command.

(In principle, the **Graphics Drawer** functions can be executed from the text interface too, but since the user isn't expected to write scripts using those functions, they aren't documented here. [1] The text versions of the graphics commands appear in log files, though. They begin with `graphics <x>`, where `<x>` is a number indicating the graphics drawer to which the command applies.)

To run OOF with the text interface, start it up with the `-text` option. You will be presented with a prompt like this:

```
OOF >>
```

All of the Menus, Functions, and Variables in the graphical interface are available in the text interface. In general, to invoke any one of them you just type its name at the prompt. All names can be *abbreviated*, as long as enough of the name is provided to distinguish it from all other legal options.

The only differences between typing commands at the keyboard and executing them from a command file are that when executing a file (a) the prompt doesn't appear when command files are read, and (b) arguments to Functions must be given explicitly (see Section 6.1.3). This Section is written as if all commands are being executed interactively.

[1] Note, however, that the graphics functions are only available when the program is connected to an X server, so they are *not* available when using the `-text` option. They *are* available when running command scripts in graphics mode.

6.1.1 Getting Help

To see a list of the available options, type a question mark:

```
OOF >> ?
=== Type "?? xxx" to get help on option "xxx".
=== Choices are (command* menu/ variable=) :
deltaT= maxiter= tolerance= globalK= preconditioner= verbose= seed=
commandfile* equilibrate* comp_equil* mutate* reset* cruise* loop*
quit* initialize/ modify/ groups/ select/ bc/ distort/ output/
graphics/ plot/ macros/ log/
OOF >>
```

A name followed by an equals sign (=) is a Variable, a name followed by an asterisk (*) is a Function, and a name followed by a slash (/) is a Menu. Typing two question marks followed by a Menu, Function, or Variable name will give more information about that specific object:

```
OOF >> ?? tolerance
tolerance = 1e-05: iterative method tolerance
OOF >> ?? distort
distort: set and apply distortions and forces
OOF >> ?? equilibrate
equilibrate: compute displacements
```

6.1.2 Menus

Typing the name of a Submenu transfers control to that Submenu, and the next command typed must be an item from the new menu. Normally, after executing a Function or Variable within a Submenu control is returned to the main OOF menu. To suppress this, use parentheses to group submenu commands (see Section 6.1.5). To escape from a Submenu without executing any other commands, type `<`.

When in a Submenu, the prompt changes to the name of the Submenu:

```
OOF >> initialize
initialize >> uniform
uniform >> <
OOF >>
```

6.1.3 Functions

Typing the name of a Function executes that function. If the function has arguments, they should be provided on the command line after the function name. Each argument must have the form `name = value`. The order of the arguments doesn't matter. Commas may be used to separate arguments, but they're optional. Here is an example from the /select/elements Menu:

```
select >> circle x=.2 y=.2 radius=.5
```

If the command is being executed interactively (*i.e,* at the keyboard, not from a command file or macro), then OOF will ask for any omitted arguments. The default value of each argument is printed in brackets; type a new value or just hit ⟨*return*⟩ to accept the default.

```
select >> circle radius=.3
*Enter x [0.2]:
*Enter y [0.2]: .5
```

The default values for the arguments are the values used the last time the command was executed.

6.1.4 Variables

Typing the name of a variable prints its current value. Typing `name = value` sets the current value. Typing `name =`, omitting the value, makes OOF prompt you for a new value. As with Function arguments, hitting ⟨*return*⟩ without entering a value retains the old value.

```
OOF >> tolerance
tolerance = 1e-05
OOF >> tolerance = 1.e-6
OOF >> tolerance =
*Enter tolerance [1e-06]: 1.e-7
```

The last form is available only during interactive operation.

6.1.5 Combining Commands

Multiple commands can be typed on one line, so you can execute Functions from Submenus like this:

```
OOF >> distort set ystrain = .01
OOF >> distort set top
OOF >> distort increment
```

Semicolons can be used to delimit commands. They are optional unless you're omitting arguments, in which case the semicolon is required so that OOF doesn't try to interpret the second command as an argument of the first one:

```
OOF >> select elements circle; modify replace empty gray=1
*Enter x [0]: 0.5
*Enter y [0]: 0.5
*Enter radius [0]: 0.2
OOF >> select elements circle modify replace empty gray=1
=== "modify" is not a recognizable argument!
=== Command "circle" not executed.
```

It's a good idea to use semicolons after all Functions and Variables when combining commands on one line.

When executing multiple commands from a single Submenu, parentheses can be used to keep OOF from popping back to the main menu after each Function or Variable. Parentheses can be nested. The first example in this section could have been written

```
OOF >> distort (
distort >> set
set >> (ystrain = .01
set >> top
set >> )
distort >> increment
distort >> )
OOF >>
```

or even

```
OOF >> distort (set (ystrain=.01; top) increment)
```

6.1.6 Defining Macros

Macros are defined by typing a series of commands enclosed in braces, followed by the name of the macro. If the name isn't supplied and you're running OOF interactively, OOF will ask you for a name. The macro will be installed as a Function in the Menu where you began the definition.

This example installs a macro called `step` in the main OOF menu:

```
OOF >> {distort increment; equilibrate} step
```

This example makes the command `select grain1` a shorthand for /select/elements/group/grain1 by installing a macro in the /select Menu:

```
select {select elements group grain1} grain1
```

Notice that the commands within the braces are interpreted as commands to the main OOF menu, not the /select menu.

The text interface can read macro files generated by the graphical interface, and vice versa.

6.1.7 Summary of Special Characters

The following characters have special meaning to the text command parser and therefore should not be used in anything that might become a command (such as an element or node group name) or a function argument.

(space) delimits words.

`;` **(semicolon)** delimits commands, ends argument lists.

`%` **(percent)** begins comments in command files.

`()` **(parentheses)** group commands within a submenu.

`""` **(quotation marks)** delimit string Function arguments containing spaces.

`?` **(question mark)** lists Menu options, or, when doubled, gets help on a specific option.

`=` **(equals)** separates Variables and Function arguments from their values.

`,` **(comma)** delimits Function arguments (optional).

`<` **(less than)** exits a Submenu.

{} **(braces)** surround macro definitions.

6.1.8 Command Line Editing

The interactive text interface in OOF uses version 1.11 of the **editline** library by Simmule Turner and Rich Salz. Here are the relevant parts of its manual:

A program that uses this library provides a simple emacs-like editing interface to its users. A line may be edited before it is sent to the calling program by typing either control characters or escape sequences. A control character, shown as a caret followed by a letter, is typed by holding down the "control" key while the letter is typed. For example, "`^A`" is a control-A. An escape sequence is entered by typing the "escape" key followed by one or more characters. The escape key is abbreviated as "`ESC`". Note that unlike control keys, case matters in escape sequences; "`ESC F`" is not the same as "`ESC f`".

An editing command may be typed anywhere on the line, not just at the beginning. In addition, a return may also be typed anywhere on the line, not just at the end.

Most editing commands may be given a repeat count, n , where n is a number. To enter a repeat count, type the escape key, the number, and then the command to execute. For example, "`ESC 4 ^f`" moves forward four characters. If a command may be given a repeat count then the text "[n]" is given at the end of its description.

The following control characters are accepted:

```
^A      Move to the beginning of the line
^B      Move left (backwards) [n]
^D      Delete character [n]
^E      Move to end of line
^F      Move right (forwards) [n]
^G      Ring the bell
^H      Delete character before cursor (backspace key) [n]
^J      Done with line (return key)
^K      Kill to end of line (or column [n])
```

```
^L     Redisplay line
^M     Done with line (alternate return key)
^N     Get next line from history [n]
^P     Get previous line from history [n]
^R     Search backward (forward if [n]) through history for text;
          prefixing the string with a caret (^) forces it to
          match only at the beginning of a history line
^T     Transpose characters
^V     Insert next character, even if it is an edit command
^W     Wipe to the mark
^X^X   Exchange current location and mark
^Y     Yank back last killed text
^[     Start an escape sequence (escape key)
^]c    Move forward to next character ``c''
^?     Delete character before cursor (delete key) [n]
```

The following escape sequences are provided.

```
ESC ^H    Delete previous word (backspace key) [n]
ESC DEL   Delete previous word (delete key) [n]
ESC SP    Set the mark (space key); see ^X^X and ^Y above
ESC .     Get the last (or [n]'th) word from previous line
ESC ?     Show possible completions; see below
ESC <     Move to start of history
ESC >     Move to end of history
ESC b     Move backward a word [n]
ESC d     Delete word under cursor [n]
ESC f     Move forward a word [n]
ESC l     Make word lowercase [n]
ESC m     Toggle if 8 bit chars display as themselves or with
             an ``M\-'' prefix
ESC u     Make word uppercase [n]
ESC y     Yank back last killed text
ESC w     Make area up to mark yankable
ESC nn     Set repeat count to the number nn
ESC C     Read from environment variable ``_C_'', where C is an
             uppercase letter
```

BUGS AND LIMITATIONS

Cannot handle lines more than 80 columns.

AUTHORS

Simmule R. Turner ⟨uunet.uu.net!capitol!sysgo!simmy⟩ and Rich Salz ⟨rsalz@osf.org⟩. Original manual page by DaviD W. Sanderson ⟨dws@ssec.wisc.edu⟩.

6.2 Control Characters Used when Setting Variables

When typing in any input box in the graphical interface, the input can be edited with the following control characters. This section is cut directly out of the XForms manual[2]:

```
delete previous char  <Delete>
delete next char      <Cntl> D
delete previous word  <Meta> <Delete>, <Cntl> W
delete next word      <Meta> d
delete to end of line <Cntl> k
backspace             <Cntl> H
to beginning of line  <Cntl> A, <Shift> <Left>
to end of line        <Cntl> E, <Shift> <Right>
char backward         <Cntl> B, <Left>
char forward          <Cntl> F, <Right>
next line             <Cntl> N, <Down>
previous line         <Cntl> P, <Up>
next page             <PageDown>
previous page         <PageUp>
word backward         <Meta> b
word forward          <Meta> f
beginning of field    <Meta> <Home>, <Shift><Up>
end of field          <Meta> <End>, <Shift><Down>
clear input field     <Cntl> U
paste                 <Cntl> y
```

You can use the mouse or the right and left arrow keys to position the cursor within the text. Double clicking on a word in the text will select that word; triple clicking will select all the text. Typed characters are inserted at the cursor, or replace the selected text.

6.3 Data File Formats

There are two file formats for `.goof` files, *ASCII* and *binary*. ASCII files are larger and slower to load, but are more portable. Binary files may not be readable on computers different than the ones on which they were created, and they cannot be examined or changed with a text editor, but they load faster and may be smaller. Furthermore, the binary files can optionally contain extra stiffness matrix information, reducing OOF's startup time (at the expense of a fair amount of disk space).

If you are writing your own `.goof` files, the ASCII format will be much easier to work with.

[2] `http://www.westworld.com/~dau/xforms/forms.html`

6.3.1 ASCII Format

The ASCII data file is read and processed by the same command parser that is used for OOF's text interface (Section 6.1). This allows some flexibility in formatting. However, it is recommended that you stick with the format described below.

An ASCII data file begins with the line

```
version number = 5
```

The version number may change in future releases. This manual describes only version 5.

The next two lines specify the number of elements and nodes in the grid.

```
Nelements = <n>
Nnodes = <n>
```

If these lines are omitted, or if the numbers are wrong, the program will still work, but memory will not be used efficiently.

There are five additional sections to the data file: nodes, elements, nodegroups, elementgroups, and startup commands. Nodes must precede elements, and both must precede the other three sections.

In what follows, quantities printed in angle brackets like <this> are values to be supplied by the user.

Nodes

Begin this section with the line:

```
nodes (
```

Each node in the grid is described by a single line of the form:

```
<name> i=<i> x=<x> y=<y> dx=<dx> dy=<dy> <etc>
```

<i> is the index of the node. Each node must have a unique non-negative integer index. The indices don't have to be in order, but all integers between 0 and $N-1$ must be present, where N is the number of nodes. <x> and <y> are the initial position of the node. <dx> and <dy> are its displacement from its initial position.

The values to be used for <name> and <etc> depend on the type of node (see Section 3.13.2).

For xy nodes, <name> is xy and <etc> is blank.

For linear nodes, <name> is linear, and <etc> is the matrix T that transforms a 2-vector in the node's coordinate system into the OOF coordinate system. T is specified as T00=<t00> T01=<t01> T10=<t10> T11=<t11>.

Close the node section with a closing parenthesis:

```
)
```

Elements

Begin this section with the line:

```
elements (
```

Each element in the grid is described by a single line of the form:

```
<name> i=<i> n1=<n1> n2=<n2> n3=<n3> gray=<g> <etc>
```

`<name>` is the name of the element type, as it appears in the menus (see Section 4.5). `<i>` is a unique element index. Like the node indices, the element indices don't have to be in order, but there can be no missing indices in the data file. `<n1>`, `<n2>`, and `<n3>` are the indices of the nodes at the corners of the element, going *counterclockwise* around the element. `<g>` is the element's gray value.

The data in `<etc>` depends on the element type. It consists of all the element's parameters, as listed in Section 4.5, in the format `<name>=<value>`. Remember that all elements have a `gray` parameter, and all but the `empty` elements have a `planestrain` parameter.

For example, an isotropic element might look like:

```
isotropic i=7 n1=4 n2=25 n3=24 gray=0.5
    poisson=0 young=1 alpha=0 planestrain=false
```

(all typed on one line, though). A hexagonal element might look like:

```
hexagonal i=252 n1=132 n2=133 n3=153 gray=0.75 orientation=[10, 90, 0]
    planestrain=false c11=1 c12=0 c13=0 c33=1 c44=0.5 alpha11=1 alpha33=1
```

and an empty element might look like:

```
empty i=533 n1=280 n2=301 n3=300 gray=1
```

Elements that have mutated (3.1.4) will have extra (as yet undocumented) parameters to describe the mutation. It is unlikely that users would want to write `.goof` files with mutated elements.

Close the element section with a closing parenthesis:

```
)
```

Node Groups

If there are any node groups, they should be listed next. The format is:

```
nodegroup (
label=<group name>
node=<i>
...
)
```

where `<i>` is the index of a node in the group. The ... means repeat the `node=` line for each node in the group.

Element Groups

Element groups can be listed before or after the node groups. The format is very similar to the node group format:

```
elementgroup (
label=<group name>
elem=<i>
...
)
```

Startup Commands

Any OOF commands at the end of the file will be executed as they are encountered. This lets boundary conditions, distortions, *etc*, be set automatically. These commands, however, must be preceded by the word `oof`, indicating that they come from the main OOF menu, and not from the menu that's used to load the nodes and elements. For example, to set fixed boundary conditions on the `top` and `bottom` node groups, put the lines

```
oof bc fix both top
oof bc fix both bottom
```

at the end of the `.goof` file. For more on the text versions of OOF commands, see Section 6.1.

6.3.2 Binary Format

The beginning and end of the binary `.goof` files are actually ASCII. The data in the middle, however, is all binary. This means that it consists of nothing but numbers written by the C `fwrite()` function. The exact order in which the numbers are written is crucial. The number of bytes used for each number in principle depends on the type of computer that's reading and writing the file, so this manual will only specify the C data type. The types and their usual sizes are

C	Fortran	size (bytes)
double	double precision	8
float	real	4
int	integer	4
char	character	1

Header

The header is all ASCII.

The file must begin with the line

```
version number = 5
```

The version number may change in future releases. This manual describes only version 5.

The next two lines specify the number of elements and nodes in the grid.

```
Nelements = <n>
Nnodes = <n>
```

If these lines are omitted, or if the numbers are wrong, the program will still work, but memory will not be used efficiently.

The next line tells OOF that this is a binary file. This line is required.

```
type = b
```

The bulk of the header consists of lists of names of element types, node types, and node and element groups. The purpose of the lists is to establish an *order* for the names, so that in the binary section of the file they can be referred to by number. The first item in each list is number 0, the second is number 1, *etc.* Each item in the lists must appear on its own line, and each list must end with a line containing only `-1`. The four lists can appear in any order, and only the first two are required.

Element Types Begin this list with the line

```
elements
```

Each element type that appears in the grid must be listed, but types that don't appear don't have to be listed. The name of the type is the name that appears in the menus (Section 4.5). For example, the list

```
empty
isotropic
-1
```

indicates that empty elements are type 0, isotropic elements are type 1, and no other element types are present.

Node Types Begin this list with the line

```
nodes
```

followed by a list of the node types used in the file. The types are xy and linear.

Node Groups Begin this list with the line

```
nodegroups
```

Element Groups Begin this list with the line

```
elementgroups
```

End the header with another `-1`, indicating that there are no more lists to follow.

For example, here is the header from the file `example2.goof` used in Section 2.2:

```
version number = 5
type = b
elements
isotropic
empty
cubic
damisotropic
hexagonal
orthorhombic
eds_el
damage
-1
nodes
xy
linear
-1
nodegroups
right
left
top
bottom
upperleft
lowerleft
upperright
lowerright
-1
elementgroups
stones
u2
rem
doors
zepplin
negativland
nirvana
stage
-1
-1
```

Binary Data

From here on, until instructed otherwise, everything in the file is binary. The data must appear in the file in the order in which they appear in the tables below. Names are provided just for reference, they aren't actually used. The sizes, order, and meaning are what matters.

Nodes

The two different node types have different representations. Both begin with an `int` type, which is the position of that type of node in the name list in the header. Because the amount of data needed for each node type is known, no special termination is needed between nodes.

XY Nodes		
type	`int`	The position of `xy` in the node name list.
flag	`char`	0 or 1. See below.
index	`int`	The node index. Must be unique, and all integers from 0 up to the number of nodes must be accounted for.
x	`float`	The node's undistorted x position.
y	`float`	The node's undistorted y position.

If flag is 1, then the node displacement is also specified:

dx	`float`	The node's x displacement.
dy	`float`	The node's y displacement.

Linear Nodes		
type	`int`	The position of `linear` in the node name list.
flag	`char`	See below.
index	`int`	The node index. Must be unique, and all integers from 0 up to the number of nodes must be accounted for.
x	`float`	The node's undistorted x position.
y	`float`	The node's undistorted y position.

If the low bit of flag is 1, then the node displacement is also specified:

dx	`float`	The node's x displacement.
dy	`float`	The node's y displacement.

If the second bit of flag is 1 (*i.e,* flag=2 or flag=3, then the transformation matrix T is also specified. T is the matrix that transforms a 2-vector in the node's coordinate system into the OOF coordinate system.

t00	`float`	The xx component of T.
t01	`float`	The xy component of T.
t10	`float`	The yx component of T.
t11	`float`	The yy component of T.

The list of nodes must end with a *binary* integer -1:

End List		
EOL	`int`	-1

Elements

Each element type has a different binary representation. This section of the file lists the data for each element. Because the amount of data for each type of element is known, no special termination is needed between elements.

Element Index Unlike the ASCII data file, the binary data file does not contain an explicit index labeling each element. Instead, the index is inferred from the order in which the elements appear in the file. The first element is number 0, the second is 1, and so on. This index is used to specify clones and element groups.

Element Type The element type is stored as an integer, indicating the position of this type in the list of element types in the header. The first element type in the list is 0, the second is 1, and so on.

Flag All elements other than the empty element have an integer flag parameter. The low bit (`0x1`) of flag is 1 if the element is in plane strain, 0 if it's in plane stress. The second bit (`0x2`) is 1 if the local stiffness matrix and thermal forces are stored. The third bit (`0x4`) is 1 if this element is a clone (see below), and the fourth bit (`0x8`) is 1 if the element has been mutated.

Clones The third bit (`0x4`) of flag is 1 if this element is a *clone*. To save space and time, elements can be flagged as being clones of other elements that have already been read. Clones must have identical geometries, orientations, and thermoelastic parameters. They can differ *only* in their node indices and gray value.[3] The big advantage of clones is that their local stiffness matrices do not have to be stored or computed; they can simply be copied from another element. The clone index appearing in the tables below is the element index of the element whose stiffness matrix is to be copied.

[3]For historical reasons, even though the parameters aren't used for clones, they must all be present in the data file. This may change in a future version.

In no case can both the clone index and the stiffness matrix K be specified at the same time. The notation "(conditional)" in the tables means that the datum in question is present only if flag is set appropriately.

Mutated Elements The fourth bit (`0x8`) of flag is 1 if the element has been mutated. The notation "(mutated)" in the tables indicates that the datum in question is present only if the element has been mutated.

Additional Data Types If stiffness matrices or thermal forces are provided in the data file, then the data type in the file below is listed as `stiffness` or `force`. The exact form of these is listed at the end of this section.

Termination The list of elements must end with a *binary* integer -1.

Data Details for Elements

The data for each element type is as follows:

Isotropic Element (Section 4.5.1)		
type	`int`	Element type.
gray	`float`	Gray value used to display the element.
node1	`int`	Index of first node, going counterclockwise around the element.
node2	`int`	Index of second node.
node3	`int`	Index of third node.
poisson	`float`	Poisson's ratio.
young	`float`	Young's modulus.
alpha	`float`	Thermal expansion coefficient.
flag	`int`	See above.
K	`stiffness`	Stiffness matrix (conditional).
F	`force`	Thermal forces (conditional).
clone	`int`	Clone index (conditional).

empty Element (Section 4.5.2)		
type	`int`	Element type.
gray	`float`	Gray value used to display the element.
node1	`int`	Index of first node, going counterclockwise around the element.
node2	`int`	Index of second node.
node3	`int`	Index of third node.

cubic Element (Section 4.5.3)		
type	`int`	Element type.
gray	`float`	Gray value used to display the element.
node1	`int`	Index of first node, going counterclockwise around the element.
node2	`int`	Index of second node.
node3	`int`	Index of third node.
L	`float`	Latitude (See Section 4.3).
R	`float`	Rotation.
S	`float`	Spin.
flag	`int`	See above.
K	`stiffness`	Stiffness matrix (conditional).
F	`force`	Thermal forces (conditional).
clone	`int`	Clone index (conditional).
poisson	`float`	Poisson's ratio.
young	`float`	Young's modulus.
alpha	`float`	Thermal expansion coefficient.
A	`float`	Anisotropy.

hexagonal Element (Section 4.5.4)		
type	`int`	Element type.
gray	`float`	Gray value used to display the element.
node1	`int`	Index of first node, going counterclockwise around the element.
node2	`int`	Index of second node.
node3	`int`	Index of third node.
L	`float`	Latitude (See Section 4.3).
R	`float`	Rotation.
S	`float`	Spin.
flag	`int`	See above.
K	`stiffness`	Stiffness matrix (conditional).
F	`force`	Thermal forces (conditional).
clone	`int`	Clone index (conditional).
c11	`float`	Elastic coefficient.
c12	`float`	Elastic coefficient.
c13	`float`	Elastic coefficient.
c33	`float`	Elastic coefficient.
c44	`float`	Elastic coefficient.
alpha11	`float`	Thermal expansion coefficient.
alpha33	`float`	Thermal expansion coefficient.

orthorhombic Element (Section 4.5.5)		
type	`int`	Element type.
gray	`float`	Gray value used to display the element.
node1	`int`	Index of first node, going counterclockwise around the element.
node2	`int`	Index of second node.
node3	`int`	Index of third node.
L	`float`	Latitude (See Section 4.3).
R	`float`	Rotation.
S	`float`	Spin.
flag	`int`	See above.
K	`stiffness`	Stiffness matrix (conditional).
F	`force`	Thermal forces (conditional).
clone	`int`	Clone index (conditional).
c11	`float`	Elastic coefficient.
c12	`float`	Elastic coefficient.
c13	`float`	Elastic coefficient.
c22	`float`	Elastic coefficient.
c23	`float`	Elastic coefficient.
c33	`float`	Elastic coefficient.
c55	`float`	Elastic coefficient.
c66	`float`	Elastic coefficient.
alpha11	`float`	Thermal expansion coefficient.
alpha22	`float`	Thermal expansion coefficient.
alpha33	`float`	Thermal expansion coefficient.

trigonalA Element (Section 4.5.7)		
type	int	Element type.
gray	float	Gray value used to display the element.
node1	int	Index of first node, going counterclockwise around the element.
node2	int	Index of second node.
node3	int	Index of third node.
L	float	Latitude (See Section 4.3).
R	float	Rotation.
S	float	Spin.
flag	int	See above.
K	stiffness	Stiffness matrix (conditional).
F	force	Thermal forces (conditional).
clone	int	Clone index (conditional).
c11	float	Elastic coefficient.
c12	float	Elastic coefficient.
c13	float	Elastic coefficient.
c14	float	Elastic coefficient.
c15	float	Elastic coefficient.
c33	float	Elastic coefficient.
c44	float	Elastic coefficient.
alpha11	float	Thermal expansion coefficient.
alpha33	float	Thermal expansion coefficient.

trigonalB Element (Section 4.5.8)		
type	`int`	Element type.
gray	`float`	Gray value used to display the element.
node1	`int`	Index of first node, going counterclockwise around the element.
node2	`int`	Index of second node.
node3	`int`	Index of third node.
L	`float`	Latitude (See Section 4.3).
R	`float`	Rotation.
S	`float`	Spin.
flag	`int`	See above.
K	`stiffness`	Stiffness matrix (conditional).
F	`force`	Thermal forces (conditional).
clone	`int`	Clone index (conditional).
c11	`float`	Elastic coefficient.
c12	`float`	Elastic coefficient.
c13	`float`	Elastic coefficient.
c14	`float`	Elastic coefficient.
c33	`float`	Elastic coefficient.
c44	`float`	Elastic coefficient.
alpha11	`float`	Thermal expansion coefficient.
alpha33	`float`	Thermal expansion coefficient.

eds_el Element (Section 4.5.6)		
type	`int`	Element type.
gray	`float`	Gray value used to display the element.
node1	`int`	Index of first node, going counterclockwise around the element.
node2	`int`	Index of second node.
node3	`int`	Index of third node.
L	`float`	Latitude (See Section 4.3).
R	`float`	Rotation.
S	`float`	Spin.
flag	`int`	See above.
K	`stiffness`	Stiffness matrix (conditional).
F	`force`	Thermal forces (conditional).
clone	`int`	Clone index (conditional).
poisson	`float`	Poisson's ratio.
young	`float`	Young's modulus.
a1	`float`	Thermal expansion coefficient.
a2	`float`	Thermal expansion coefficient.
a3	`float`	Thermal expansion coefficient.

damisotropic Element (Section 4.5.9)		
type	`int`	Element type.
gray	`float`	Gray value used to display the element.
node1	`int`	Index of first node, going counterclockwise around the element.
node2	`int`	Index of second node.
node3	`int`	Index of third node.
poisson	`float`	Poisson's ratio.
young	`float`	Young's modulus.
alpha	`float`	Thermal expansion coefficient.
max_s	`float`	Maximum stress.
min_s	`float`	Minimum stress.
max_kd	`float`	Maximum knockdown factor.
min_kd	`float`	Minimum knockdown factor.
flag	`int`	See above.
K	`stiffness`	Stiffness matrix (conditional).
F	`force`	Thermal forces (conditional).
clone	`int`	Clone index (conditional).
total_kd	`float`	Product of all applied knockdowns (mutated.)
oldgray	`float`	Gray value before mutation (mutated).

damage Element (Section 4.5.10)		
type	int	Element type.
gray	float	Gray value used to display the element.
node1	int	Index of first node, going counterclockwise around the element.
node2	int	Index of second node.
node3	int	Index of third node.
L	float	Latitude (See Section 4.3).
R	float	Rotation.
S	float	Spin.
flag	int	See above.
K	stiffness	Stiffness matrix (conditional).
F	force	Thermal forces (conditional).
clone	int	Clone index (conditional).
poisson	float	Poisson's ratio.
young	float	Young's modulus.
alpha	float	Thermal expansion coefficient.
max_s	float	Maximum stress.
kd_1	float	First knockdown factor.
kd_2	float	Second knockdown factor.
only_once	int	Mutate only once?
oldgray	float	Gray value before mutation (mutated).
C_ijkl	stiffness	Mutated 3-D stiffness matrix (mutated).

griffith and griffith2 Elements (Section 4.5.11)		
type	`int`	Element type.
gray	`float`	Gray value used to display the element.
node1	`int`	Index of first node, going counterclockwise around the element.
node2	`int`	Index of second node.
node3	`int`	Index of third node.
L	`float`	Latitude (See Section 4.3).
R	`float`	Rotation.
S	`float`	Spin.
flag	`int`	See above.
K	`stiffness`	Stiffness matrix (conditional).
F	`force`	Thermal forces (conditional).
clone	`int`	Clone index (conditional).
poisson	`float`	Poisson's ratio of unmutated element.
young	`float`	Young's modulus of unmutated element.
alpha	`float`	Thermal expansion coefficient.
gamma	`float`	Surface energy density.
kd_1	`float`	First knockdown factor.
kd_2	`float`	Second knockdown factor.
only_once	`int`	Mutate only once?
oldgray	`float`	Gray value before mutation (mutated).
C_ijkl	`stiffness`	Mutated 3-D stiffness matrix (mutated).

zimmer and zimmer2 Elements (Section 4.5.13)		
type	int	Element type.
gray	float	Gray value used to display the element.
node1	int	Index of first node, going counterclockwise around the element.
node2	int	Index of second node.
node3	int	Index of third node.
L	float	Latitude (See Section 4.3).
R	float	Rotation.
S	float	Spin.
flag	int	See above.
K	stiffness	Stiffness matrix (conditional).
F	force	Thermal forces (conditional).
clone	int	Clone index (conditional).
c11	float	Elastic coefficient.
c12	float	Elastic coefficient.
c13	float	Elastic coefficient.
c33	float	Elastic coefficient.
c44	float	Elastic coefficient.
alpha11	float	Thermal expansion coefficient.
alpha33	float	Thermal expansion coefficient.
gamma	float	Surface energy density.
kd_1	float	First knockdown factor.
kd_2	float	Second knockdown factor.
only_once	int	Mutate only once?
oldgray	float	Gray value before mutation (mutated).
C_ijkl	stiffness	Mutated 3-D stiffness matrix (mutated).

The local stiffness matrix is a 6×6 symmetric matrix K, giving the element's contribution to the global stiffness K_{global}. It's easiest by far to leave it out of the file. If you are writing your own binary `.goof` file, and really really want to include the stiffness matrices, it's best to write the file without them, load it into OOF and use /output/grid/binary (Section 3.31.2) with save_stiffness = `true` to write a new `.goof` file. For completeness, though, here is the binary format used to save the local stiffness matrix and thermal forces:

Stiffness Matrices		
k00	double	K_{00}: only the lower triangular part of K is stored.
k10	double	K_{10}
k11	double	K_{11}
k20	double	K_{20}
k21	double	K_{21}
k22	double	K_{22}
k30	double	K_{30}
k31	double	K_{31}
k32	double	K_{32}
k33	double	K_{33}
k40	double	K_{40}
k41	double	K_{41}
k42	double	K_{42}
k43	double	K_{43}
k44	double	K_{44}
k50	double	K_{50}
k51	double	K_{51}
k52	double	K_{52}
k53	double	K_{53}
k54	double	K_{54}
k55	double	K_{55}

Thermal Forces		
F0x	double	x component of thermal force on node 0.
F0y	double	y component of thermal force on node 0.
F1x	double	x component of thermal force on node 1.
F1y	double	y component of thermal force on node 1.
F2x	double	x component of thermal force on node 2.
F2y	double	y component of thermal force on node 2.

Node Groups

Each of the node groups is listed after the elements, in the order in which they appear in the node group name list in the header. Each list is a sequence of integers, giving the indices of the nodes in the group. Each list must end with a binary integer -1.

Node Groups		
`n1`	`int`	Index of first node in first node group.
…		
`nn`	`int`	Index of last node in first node group.
`EOL`	`int`	-1
`n1`	`int`	Index of first node in second node group.
…		*etc.*

Element Groups

The element groups are written in an analogous way to the node groups. The groups must be listed in the order in which they appear in the element group name list in the header. Each list is a sequence of integers giving the element indices of the elements in the group. Each list must end with a binary integer -1.

Element Groups		
`e1`	`int`	Index of first element in first element group.
…		
`en`	`int`	Index of last element in first element group.
`EOL`	`int`	-1
`e1`	`int`	Index of first element in second element group.
…		*etc.*

Startup Commands

This final section of the binary data file is again in ASCII. It is identical to the Startup Commands section of the ASCII data file: a series of arbitrary OOF commands, each preceded by the word `oof`.

6.4 Known Bugs

6.4.1 Important Bugs

OOF doesn't check to see if there is sufficient memory on the X server to allocate the pixmaps used in the Drawers. If the image gets too big, the server will either start to swap or die with a `BadAlloc` error. The image can get too big for at least two reasons: the user can zoom in too far, or points can run off to infinity during the equilibration of a badly defined system (one with no fixed nodes, for example).

When selecting nodes in the graphics window, shift-clicking on a selected node unselects it, but does not redraw it, so it still appears to be selected. Use the **Redraw** button in the **Coordinates** Dashboard to see the correct set of selected nodes.

6.4.2 Less Important Bugs

Log files can contain duplicate lines. Macros loaded from the `.oofrc` file are saved in the log file. In a later session, if this log file is read with the `-file` command line option or loaded with /commandfile, the macro will be defined twice: once in `.oofrc` and once in the log file. The macro definition will appear *twice* in a log of this session. This isn't a big problem, but the log files will keep growing.

Other bugs may be listed on the OOF web site.

Acknowledgments

OOF is a big project. We are grateful for, and continue to rely on, help from a number of people.

Roldan Pozo provided invaluable advice on C++ and sparse matrix methods. Andy Roosen helped with the initial design and is our personal wizard.

Pioneers who bravely tested early versions of OOF include Jill Glass, Chun-Hway Hsueh, Thomas Isabell, Mark Knackstedt, James McMahon, Anil Saigal, Toyce Small, Mike Zimmerman, and André Zimmermann. Their comments and suggestions are much appreciated.

OOF is supported by the Center for Theoretical and Computational Materials Science[4] in the Materials Science and Engineering Laboratory[5] and by the Mathematical and Computational Sciences Division[6] of the Information Technology Laboratory[7] at the National Institute of Standards and Technology[8].

[4]http://www.ctcms.nist.gov/
[5]http://www.msel.nist.gov/
[6]http://math.nist.gov/mcsd/
[7]http://www.itl.nist.gov/
[8]http://www.nist.gov/

Bibliography

[1] Stephen A. Langer, W. Craig Carter, and Edwin R. Fuller. The ppm2oof manual. to be published as a NIST technical report.

[2] J.F. Nye. *Physical Properties of Crystals*. Oxford University Press, Oxford, 1985.